Computer Arithmetic for Probability Distribution Variables

Weiye Li *, James Mac Hyman

Mathematical Modeling and Analysis, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Abstract

The uncertainty in the variables and functions in computer simulations can be quantified by probability distributions and the correlations between the variables. We augment the standard computer arithmetic operations and the interval arithmetic approach to include probability distribution variable (**PDV**) as a basic data type. Probability distribution variable is a random variable that is usually characterized by generalized probabilistic discretization. The correlations or dependencies between PDVs that arise in a computation are automatically calculated and tracked. These correlations are used by the computer arithmetic rules to achieve the convergent approximation of the probability distribution function of a PDV and to guarantee that the derived bounds include the true solution. In many calculations, the calculated uncertainty bounds for PDVs are much tighter than they would have been had the dependencies been ignored. We describe the new PDV Arithmetic and verify the effectiveness of the approach to account for the creation and propagation of uncertainties in a computer program due to uncertainties in the initial data.

Key words: PDV arithmetic; Probabilistic arithmetic; Interval arithmetic; Interval computation; Epistemic uncertainty; Uncertainty computation; Monte Carlo; Probabilistic discretization; Probability bound; Probability box; Dependency bound; Dependency tracking

* Corresponding author. Group T-7, Mail Stop B284, Los Alamos National Laboratory, Los Alamos, NM 87545, USA; Tel.: +1-505-667-1842; Fax: +1-505-665-5757

Email addresses: `liw@lanl.gov` (Weiye Li), `mac@t7.lanl.gov` (James Mac Hyman).

1 Introduction

Computational uncertainties are unavoidable in numerical calculations. The generation and propagation of uncertainties in the initial conditions, data and the constants in mathematical model can have serious implications in the reliability of the simulation and the decisions being made based on the simulation. These inherent uncertainties cannot be made arbitrarily small by additional computations, using higher order formulas, or carrying more significant digits in the arithmetic.

Even when one has a good grasp on the accuracy of the initial data for a computation (such as measured data from a well-calibrated experiment) often little is known a priori about the accuracy of the results. The uncertainties in a simulation can pollute the reliability of the results at every stage of a computation. They may grow or even shrink during the calculations. The magnitude of computational uncertainty is of great concern to any thoughtful programmer, since no computation can be considered complete unless one has some knowledge of its accuracy.

When the accuracy of the final result is crucial and the computation is done in ordinary arithmetic, then it must be tested by repeated tedious calculations or sensitivity analysis. Approaches and representations of uncertainty within computer simulations must be developed that are efficient and provide sharp realistic bounds for the propagation of uncertainty. Analytical techniques are ideal in getting close forms of solutions while very few real-world problems can be solved by these techniques. Numerical methods become widely used. Monte Carlo [42] is a powerful approach, but has some serious shortcomings [13], such as difficulties in handling uncertainties that have unknown dependency relationships or that are with imprecise probabilities, that is, with distributions that are not fully specified.

Non-Monte Carlo methods have been developed since 1960s [19]. In the early algorithms [8, 19, 21, 35], independent relationships were assumed among all the random variables and no dependency issues were taken into account throughout a computation. Later copula based approach, which is based on the theory of copulas [36], was studied. This approach is focused on finding the bounds for joint distributions from their given marginal distributions, when the dependency relationships among the random variables are unknown. Some early work in this approach was done by Frank et al. [16] for bounding sums of random variables. Then it was extended by Williamson in his Ph.D. dissertation [45] and Downs [46] to bounding the results of adding, subtracting, multiplying and dividing random variables. More recent work of Cossette et al. [9] provided results on multivariate copula models. Ferson's RAMAS Risk Calc [14] as part of a commercial software package is an implementation of the copula approach.

Another major non-Monte Carlo based approach is interval approach. It relies on probabilistic discretizations of random variables. Berleant et al.'s DEnv algorithm [2–5] is a recent representative of this approach. DEnv makes use of linear programming [22] to achieve dependency bounds for random variables that may be independent, have unknown dependency, or have a correlation value as limited information about the dependency. Statool [43] implementing DEnv is a tool for operations on distribution functions and p-boxes [15].

Regen et al. [39] show that DEnv and the copula-based Probabilistic Arithmetic [45, 46] are equivalent in important ways. Although both of these algorithms can draw sharp bounds for joint distributions from given marginal distributions with unknown dependency, neither approach tracks the dependencies that develop between the random variables throughout a computation. Consequently in multi-step calculations, these approaches assume that the new dependency relationships of the marginal distributions obtained from the previous step of calculation are either independent or unknown. In computer programs where many of the random variables have strong deterministic function relationships, this assumption can quickly cause the bounding intervals to become useless over estimates of the sharp bounds. Berleant et al. [4, 6] describes how DEnv deals with correlation between input distributions to obtain tighter bounds using correlation coefficient as a known interval. However, these correlations need to be automatically updated in the next step of calculation and the mathematical theory validated for giving sharp bounds must still be developed.

Therefore new approach that can perform dependency tracking has to be developed. Using generalized probabilistic discretizations of probability distribution variables (PDVs), which are equivalent to random variables, we have developed a new computer arithmetic. Our *PDV Arithmetic* extends an interval approach with dependency tracking [18] and is closely related to the extensive advances in interval arithmetic [34].

PDV Arithmetic is distinguished from the other existing approaches in a sense that it tracks the dependencies that develop throughout a computation and uses this information to obtain tight bounds that are guaranteed to converge to the sharp bounds as the input generalized probabilistic discretizations are refined. Thus, it can account for the creation and propagation of uncertainties in a computer program due to uncertainties in the data and that dynamically arise in the computation. PDV Arithmetic also provides convergent approximations of the exact generalized probabilistic discretizations of PDVs based on the input random variables. The exact bounds for distributions of PDVs can be derived directly from the exact generalized probabilistic discretizations with respect to the input random variables.

PDV Arithmetic can also be applied in interval computations by ignoring the probability part in the arithmetic. With the dependency tracking feature, it gives sharp bounds to the results of any interval functions that are in form of algebraic expressions.

The current version of PDV Arithmetic assumes that all PDV inputs are independent. Although this assumption requires some additional work to define any known correlations between the input random variables to minimized the number of the inputs such that the inputs are independent, it greatly simplifies the analysis and allows us to lay a foundation for later versions of PDV Arithmetic that will allow for general dependency relationships among input random variables.

We implement PDV Arithmetic by adding a new data type, called probability distribution variable (PDV), to an existing computer language. This follows the approaches of several interval arithmetic implementations [23,25] where the programmer can embed interval arithmetic in existing complex codes. Also, many of the technical details can be easily hidden from the program with the help of a preprocessing program that converts the extended language into a standard portable version of the program.

The structure of this paper is as follows. We first introduce some concepts needed to define the PDV arithmetic rules. These include the definition of PDV, the independence and correlation among PDVs, and the generalized probabilistic discretizations of PDVs. We then describe and analyze the underlying algorithm that defines PDV Arithmetic. Next we give a brief description of how we implement PDV Arithmetic in Fortran 77 by using a Perl preprocessor and subroutine library of PDV arithmetic operations. We use two sets of numerical examples to illustrate how PDV Arithmetic automatically characterizes the uncertainties in calculating the eigenvalues of a matrix of PDVs and the challenge problem set proposed by Oberkampf et al. [38].

2 Basic Concepts and Results

In this section, we introduce some concepts and results without giving proofs. Mathematical details can be found in [31].

Definition 2.1 *(Probability Distribution Variable)*
A Probability Distribution Variable (PDV) *is a random variable [12]. The range interval of a PDV is the smallest closed interval that contains the support [12] of the PDV.*

Definition 2.2 *(Independence)*
A number of PDVs x_1, ..., x_n are independent *if they are independent random variables [12], i.e., for any Borel sets A_1, ..., A_n on the real line,*

$$Prob\{x_{i_1} \in A_{i_1}, \dots, x_{i_k} \in A_{i_k}\} = \prod_{j=1}^{k} Prob\{x_{i_j} \in A_{i_j}\}\,,$$

where $1 \le k \le n$, $1 \le i_1 < \dots < i_k \le n$.

Definition 2.3 *(Pre-image PDV Set)*
If PDV x is a function f of PDVs e_1, e_2, ..., e_l where f is not constant at any e_i when the others are fixed, $1 \le i \le l$, then set $\{e_1, e_2, ..., e_l\}$ is called a pre-image PDV set *of x.*

Definition 2.4 *(Correlation)*
Two PDVs are dependent *if they have common pre-image PDV set. Two PDVs are* partially dependent *if their pre-image PDV sets have nonempty intersection. Both dependent and partially dependent are called* correlated.

Note: Every two PDVs in a computer program must be dependent, partially dependent, or independent if the input PDVs are independent.

Definition 2.5 *(Generalized Probabilistic Discretization)*
A generalized probabilistic discretization *(GPD) of PDV x is defined by*

$$GPD(x) = \Big\{(I_r, p_r) \,\Big|\, 1 \le r \le l\,, \sum_{r=1}^{l} p_r = 1\,, \; I_r \text{ is an interval}\,, \; \text{the support of } x \subseteq \bigcup_{r=1}^{l} I_r \Big\}\,,$$

where I_r is called a bin and p_r is the associated probability with which x assumes values in I_r. The width of $GPD(x)$ is defined by

$$\|GPD(x)\| = \sup_{1 \le r \le l} \{\|I_r\|\}\,.$$

In particular, if intervals I_r's are mutually disjoint, then the above $GPD(x)$ is called a probabilistic discretization *of x, denoted by $PD(x)$.*

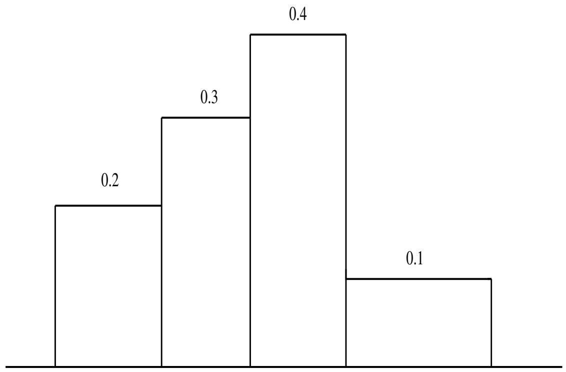

Fig. 2.1. Illustrative figure of the probabilistic discretization (PD) of a PDV with 4 bins and probability distribution $\{0.2, 0.3, 0.4, 0.1\}$ in the range interval.

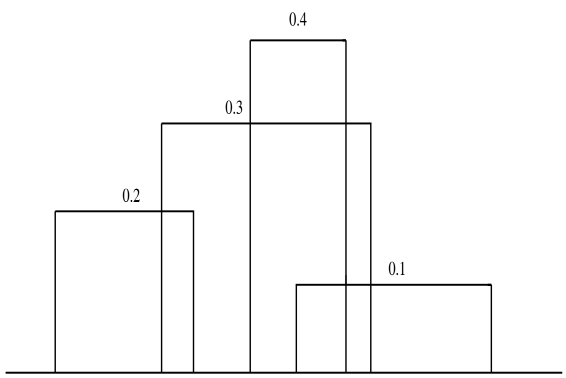

Fig. 2.2. Illustrative figure of the generalized probabilistic discretization (GPD) of a PDV with 4 bins and probability distribution $\{0.2, 0.3, 0.4, 0.1\}$ in the range interval.

Examples of probabilistic discretization and generalized probabilistic discretization are illustrated in Figures 2.1 and 2.2. Notice that probabilistic discretization is graphed as histogram.

Definition 2.6 *(Refinement of Generalized Probabilistic Discretization)*
Assume that $GPD_1(x)$ and $GPD_2(x)$ are two generalized probabilistic discretizations of PDV x. $GPD_2(x)$ is a refinement *of $GPD_1(x)$ if $GPD_2(x)$ has the following representation*

$$GPD_2(x) = \left\{ (I_r,\, p_r) \,\middle|\, r = 1, \ldots, l_{(I,p)};\ \bigcup_{r=1}^{l_{(I,p)}} I_r = I,\ \sum_{r=1}^{l_{(I,p)}} p_r = p\,;\ \forall\,(I,p) \in GPD_1(x) \right\},$$

where I_r are intervals.

Note: The refinements of intervals I allow overlaps among the interval pieces I_r.

Definition 2.7 *(Inclusion of Generalized Probabilistic Discretization)*
Assume that $GPD_1(x)$ and $GPD_2(x)$ are two generalized probabilistic discretizations of PDV x. $GPD_2(x)$ is included *in $GPD_1(x)$, denoted by $GPD_2(x) \subseteq GPD_1(x)$, if there exists a one-to-one correspondence $\pi : (I_2, p_2) \in GPD_2(x) \longmapsto (I_1, p_1) \in GPD_1(x)$ such that $I_2 \subseteq I_1$, and $p_2 = p_1$.*

Note: The inclusion of generalized probabilistic discretization can be viewed as a special refinement of generalized probabilistic discretization by considering that I_1 is refined into

$$I_1 = I_2 \bigcup (I_1 \backslash I_2)$$

and $I_1 \backslash I_2$ is assigned with zero probability.

Definition 2.8 *(Convergence of Generalized Probabilistic Discretization)*
Assume that $GPD(x)$ and $GPD_r(x)$ ($r \in \Lambda$, Λ is an index set) are generalized probabilistic discretizations of PDV x. Assume that r_0 is a limit point of Λ. $GPD_r(x)$ is convergent *to $GPD(x)$ as r tends to r_0, denoted by $GPD_r(x) \to GPD(x)$ as $r \to r_0$, or $\lim_{r \to r_0} GPD_r(x) = GPD(x)$, if there exists a one-to-one correspondence $\pi_r : (I_r, p_r) \in GPD_r(x) \longmapsto (I, p) \in GPD(x)$ for each $r \in \Lambda$, such that $a_r \to a$, $b_r \to b$, and $p_r = p$ as $r \to r_0$, where a, b and a_r, b_r are the left and right endpoints of I and I_r, respectively.*

Theorem 2.1 *(Relationship between a PDV and its Generalized Probabilistic Discretizations)*
Each generalized probabilistic discretization of PDV x corresponds to a class of PDVs that have the same generalized probabilistic discretization and converge pointwise to x everywhere when the width of the generalized probabilistic discretization tends to zero.

Proof: See [31].

Note: Generalized probabilistic discretization of PDV corresponds to the *basic probability assignment* in Evidence Theory [27]. Theorem 2.1 provides a way to group PDVs by their generalized probabilistic discretization. Each generalized probabilistic discretization corresponds to a family of PDVs, and each PDV in the family is a representative of the family.

Consider a PDV x where

$$GPD(x) = \{(I_r, p_r) \,\big|\, 1 \leq r \leq l\} .$$

Assume that the left endpoint and right endpoint of I_r are a_r and b_r, respectively.

Define two probability distribution functions $M_{GPD(x)}$ and $m_{GPD(x)}$ as the step functions:

$$\begin{aligned} M_{GPD(x)}(z) &= \sum_{r \,|\, a_r \leq z} p_r \,, \\ m_{GPD(x)}(z) &= \sum_{r \,|\, b_r \leq z} p_r \,, \quad z \in \mathbb{R} . \end{aligned}$$

We denote the class of PDVs determined by $GPD(x)$ as $C_{GPD(x)}$.

Theorem 2.2 *Let F_x be the distribution function of PDV x and F_y be the distribution function of PDV $y \in C_{GPD(x)}$. Then*

$$m_{GPD(x)}(z) \leq F_y(z) \leq M_{GPD(x)}(z) \,, \qquad \forall z \in \mathbb{R}, \quad \forall y \in C_{GPD(x)} .$$

Furthermore,

$$\begin{aligned} F_x(z) &= \lim_{\|GPD(x)\| \to 0} M_{GPD(x)}(z) \\ &= \lim_{\|GPD(x)\| \to 0} m_{GPD(x)}(z) \,, \quad z \in \mathbb{R} . \end{aligned}$$

Proof: See [31].

The graphs of $M_{GPD(x)}$ and $m_{GPD(x)}$ are disconnected since $M_{GPD(x)}$ and $m_{GPD(x)}$ are step functions. If we connect the disconnecting points with vertical lines, the graphs become connected and we call them the *connected graphs* of $M_{GPD(x)}$ and $m_{GPD(x)}$, respectively. Notice that these two graphs have the same starting and ending points.

Theorem 2.2 supports the following definition.

Definition 2.9 *(Probability Distribution Bounds, Probability Distribution Box, Refined Probability Distribution Box)*
$M_{GPD(x)}$ and $m_{GPD(x)}$ are called the upper and lower probability distribution bounds *(upper and lower p-bounds) of x with respect to $GPD(x)$, respectively. The area enclosed by the connected graphs of $M_{GPD(x)}$ and $m_{GPD(x)}$ is called the* probability distribution box *(p-box) of x with respect to $GPD(x)$. If $GPD_2(x)$ is a refined generalized probabilistic discretization of x with respect to $GPD_1(x)$, the p-box determined by $GPD_2(x)$ is called a* refined probability distribution box *(refined p-box) of x with respect to $GPD_1(x)$.*

Note: The above defined probability distribution bounds are related to the belief and plausibility measures in Evidence Theory (Dempster and Shafer [10, 11, 41]). The term "p-box" was first used by Ferson [15].

Definition 2.10 *(Interval Arithmetic)* *Interval arithmetic is a set of rules for set operations defined on intervals and is based on algebraic expressions, called* base functions. *Each variable in base function expression is replaced with the interval to which this variable belongs. These replacement intervals are considered independent regardless whether some of them replace the same variable. Equivalently, the derived set from the arithmetic is the image of a function called* derived function, *which is derived from the base function by replacing each appearance of a variable in the algebraic expression of the base function with a new different variable symbol, over the Cartesian product of the intervals that these new variables belong to.*

Theorem 2.3 *(Connectedness, Convergence and Continuity of Interval Arithmetic)* *If f is a continuous base function over the Cartesian product of intervals I_1, ..., I_n, then the derived set from using the interval arithmetic, denoted by $f[I_1, ..., I_n]$, is an interval. Furthermore, if f is continuous over the Cartesian product of the closures of I_1, ..., I_n, then the width of $f[I_1, ..., I_n]$ tends to zero as the widths of I_1, ..., I_n tend to zero. If f is continuous over the Cartesian product of intervals $I_1, ..., I_n$ as well as the Cartesian product of intervals $\tilde{I}_1$, ..., $\tilde{I}_n$, then $f[\tilde{I}_1, ..., \tilde{I}_n] \to f[I_1, ..., I_n]$ as $\tilde{I}_i \to I_i$ ($i = 1, ..., n$), provided one of the following two conditions is satisfied:*

(1) I_1, ..., I_n are finite intervals.
(2) $\tilde{I}_1 \subseteq I_1$, ..., $\tilde{I}_n \subseteq I_n$.

Proof: See [31].

Remark: The interval arithmetic in Definition 2.10 is also called *standard interval arithmetic* by Williamson [45] and *idealized interval arithmetic* by Kearfott [24]. The "derived function" concept here is consistent with the term "interval extension" in Moore [33, 34]. The notation $f[I_1, ..., I_n]$ that we use here is the image of the interval extension over intervals I_1, ..., I_n. It is true that $f(I_1, ..., I_n) \subseteq f[I_1, ..., I_n]$ [33, 34]. For instance, let $f(x) = x - x$, and $x \in I = [0, 1]$. Then $f(I) = \{0\}$, while $f[I] = [0, 1] - [0, 1] = [-1, 1]$.

A form of interval arithmetic first appeared at least as early as in 1924 [7] and 1931 [47], then in 1958 [44]. Modern development of interval arithmetic began with Moore's dissertation [32] in 1962. Since then there have been thousands of publications in the field of interval analysis and its applications, plus an increasing amount of software support for interval computations. Major advances have been made by Moore [33,34], Nickel [37], Alefeld and Herzberger [1], Kearfott [24,25] and Kreinovich [26], Kreinovich et al. [29], and Jaulin et al. [20]. A web site for interval computations is
`http://www.cs.utep.edu/interval-comp/`.

3 Algorithm of PDV Arithmetic

PDV Arithmetic is a set of rules to calculate generalized probabilistic discretizations of PDVs. A PDV can be a discrete, continuous, or mixed random variable. We store a PDV in computer by its generalized probabilistic discretization. In PDV Arithmetic, the calculations among PDVs are actually the calculations among the generalized probabilistic discretizations of PDVs. Theorem 2.1 tells us that if the width of the generalized probabilistic discretization of each PDV is small, we may obtain precise results for the PDVs. Therefore, *the task of PDV Arith-*

metic is to find generalized probabilistic discretization for each PDV whose width is small when the widths of the generalized probabilistic discretizations of the input PDVs are small.

We classify two types of PDVs in PDV Arithmetic: *primitive PDV* and *derived PDV*. Primitive PDV is explicitly defined as a PDV input by the user. It can also be called *input PDV*. Derived PDV is defined as a function of primitive PDV(s) via algebraic calculations. Derived PDV includes intermediate PDV, which is not declared in the source program as PDV but arises in the intermediate stages of a computation.

Primitive PDV is merely a PDV input. Although it is defined via a variable name, it is independent of the variable name. On the contrary, the variable name via which a primitive PDV is defined is the identity function of the primitive PDV and is a derived PDV. From this point of view, *all PDV names in a program are functions of the PDV inputs and hence are derived PDVs.*

Primitive PDVs control the relationships among derived PDVs generated during the execution of the program. Thus derived PDVs are like puppets on a stage connected through interrelated strings governed by primitive PDVs.

There are cases when primitive PDVs are correlated. If the correlations of the primitive PDVs can be expressed as function relations or can be approximated by algebraic expressions from statistical regressions [17], then some primitive PDVs that are functions of other primitive PDVs can be converted to derived PDVs. This approach can reduce the number of correlated primitive PDVs to the maximal extend and improve the efficiency of the program.

In the current version of PDV Arithmetic, we require the following condition.

Condition 3.1 *All primitive PDVs are independent.*

Condition 3.1 provides mathematical convenience to develop PDV Arithmetic. Although this assumption excludes problems with correlated input random variables that cannot be reduced to independent primitive PDVs, using the above approach there are still many problems where the condition is satisfied.

To satisfy Condition 3.1, PDV inputs regardless whether they are defined by the same variable names or have identical generalized probabilistic discretizations, are considered different primitive PDVs. This guarantees that every derived PDV has a non-ambiguous pre-image PDV set. That is, if a PDV is redefined using the same variable name, then it is treated as a new, and different, primitive PDV. *Thus redundant definitions of the same PDV in a computer program should be avoided, unless the programmer means to define a different primitive PDV.* The following example (written in pseudo codes) illustrates how this rule works.

$$PDV \quad x, y, z, w$$

$$GPD(x) = \{([1,3], 0.7); ([2,4], 0.3)\}$$

$$y = x^3$$

$$GPD(x) = \{([1,3], 0.7); ([2,4], 0.3)\}$$

$$z = x^3$$

$$w = y - z$$

x is used twice as a variable symbol to input PDV data. Although y, z and w are derived from the same variable symbol x, their *pre-image primitive PDV sets* are different. y is derived from the first PDV input (first primitive PDV) only, z is derived from the second PDV input (second primitive PDV) only, and w is derived from the first and second PDV inputs (first and second primitive PDVs). Notice that both y and z have identical but independent generalized probabilistic discretizations. Hence there is *no* such a result that $w = y - z$ is equal to 0 with probability 1.

The above rule is summarized in the following condition that is also required in PDV Arithmetic.

Condition 3.2 *Each PDV input defines a new primitive PDVs.*

When calculation between two PDVs occur, it is passed to the bins and the probabilities in the generalized probabilistic discretizations. When the two PDVs are independent, any arbitrary pair of bins from the two generalized probabilistic discretizations can be used to perform the calculation. When dependency or partial dependency exists in the two PDVs, only some pairs of bins from the two generalized probabilistic discretizations can be used to perform the calculation. These pairs are determined by the common primitive PDVs in the two pre-image primitive PDV sets. After the possible pair of bins are determined, the interval arithmetic in Definition 2.10 is utilized to obtain a generalized probabilistic discretization.

In PDV Arithmetic, a bin of a derived PDV is distinguished from other bins by a set of indices of the bins (from which the bin is determined) of the primitive PDVs in the pre-image primitive PDV set of this PDV. Any non-input PDVs are derived from functions that are algebraic expressions of other PDVs and these functions can be decomposed into a sequence of unitary or binary operations on the involved PDVs. The cases in which a PDV z is derived from a PDV x only (unitary operation), or from PDVs x and y (binary operation), are analyzed as follows.

(1) *z is a function of x only.*
Suppose that x is a function of primitive PDVs e_1, ..., e_r. Each bin of x with indices $(i_{e_1}, ..., i_{e_r})$ and with the corresponding probability $p_x(i_{e_1}, ..., i_{e_r})$ determines a bin of z with indices $(i_{e_1}, ..., i_{e_r})$ and with the corresponding probability $p_z(i_{e_1}, ..., i_{e_r}) = p_x(i_{e_1}, ..., i_{e_r}) = \prod_{j=1}^{r} p_{e_j}(i_{e_j})$.

(2) *z is derived from x and y.*
   (a) *x and y are independent.*
   Suppose that x is a function of primitive PDVs e_1, ..., e_r, and y is a function of primitive PDVs e_{r+1}, ..., e_{r+s}. By independence, any arbitrary pair of bins of x and y can form a bin of z via the interval arithmetic and via multiplying the corresponding probabilities. That is, the bin of x with indices $(i_{e_1}, ..., i_{e_r})$ and with the corresponding probability $p_x(i_{e_1}, ..., i_{e_r})$ and the bin of y with index $(i_{e_{r+1}}, ..., i_{e_{r+s}})$ and with the corresponding probability $p_y(i_{e_{r+1}}, ..., i_{e_{r+s}})$ form the bin of z with indices $(i_{e_1}, ..., i_{e_{r+s}})$ and with the corresponding probability $p_z(i_{e_1}, ..., i_{e_{r+s}}) = p_x(i_{e_1}, ..., i_{e_r}) \cdot p_y(i_{e_{r+1}}, ..., i_{e_{r+s}}) = \prod_{j=1}^{r+s} p_{e_j}(i_{e_j})$.
   (b) *x and y are dependent.*
   Suppose that x and y are functions of primitive PDVs e_1, ..., e_r. Only the bin of x and the bin of y with common indices are eligible to form a bin of z, and the corresponding probabilities must be equal. That is, the bin of x and the bin of y with common indices $(i_{e_1}, ..., i_{e_r})$ form the bin of z with indices $(i_{e_1}, ..., i_{e_r})$ and with the corresponding probability

$p_z(i_{e_1}, ..., i_{e_r}) = p_x(i_{e_1}, ..., i_{e_r}) = p_y(i_{e_1}, ..., i_{e_r}) = \prod_{j=1}^{r} p_{e_j}(i_{e_j})$.

(c) *x and y are partially dependent.* Suppose that x is a function of primitive PDVs e_1, ..., e_r, e_{r+1}, ..., e_{r+s}; and y is a function of primitive PDVs e_{r+1}, ..., e_{r+s}, e_{r+s+1}, ..., e_{r+s+t}, where $r \geq 0$, $s > 0$, $t \geq 0$. Only the bin of x and the bin of y that have common values on $i_{e_{r+1}}, ..., i_{e_{r+s}}$ in the two index sets $(i_{e_1}, ..., i_{e_{r+s}})$ and $(i_{e_{r+1}}, ..., i_{e_{r+s+t}})$ can be used to form a bin of z. That is, the bin of x with indices $(i_{e_1}, ..., i_{e_{r+s}})$ and with the corresponding probability $p_x(i_{e_1}, ..., i_{e_{r+s}})$ and the bin of y with indices $(i_{e_{r+1}}, ..., i_{e_{r+s+t}})$ and with the corresponding probability $p_y(i_{e_{r+1}}, ..., i_{e_{r+s+t}})$ form the bin of z with indices $(i_{e_1}, ..., i_{e_{r+s+t}})$ and the corresponding probability $p_z(i_{e_1}, ..., i_{e_{r+s+t}}) = \prod_{j=1}^{r+s+t} p_{e_j}(i_{e_j})$.

Note: Although the decomposition of function f into a sequence of unitary or binary operations is not unique, different decompositions of f applied on I_1, ..., I_n using the interval arithmetic end up with the same derived interval $f[I_1, ..., I_n]$ (see [31]).

The algorithm described as above defines an arithmetic called *Primitive PDV Arithmetic* that is based on primitive PDVs.

Definition 3.1 *(Primitive PDV Arithmetic)*
Let $\{e_1, ..., e_n\}$ be the pre-image primitive PDV set of PDV x, and $x = f(e_1, ..., e_n)$ where f is a continuous algebraic expression on the Cartesian products of the bins of $e_1, ..., e_n$. f can be decomposed into a sequence of unitary or binary operations. Primitive PDV Arithmetic is a set of rules, which are described in the above cases 1, 2(a), 2(b), and 2(c) for any decomposition of f, to determine a generalized probabilistic discretization of x. Although the decomposition of f into a series of unitary and binary operations is not unique, the generalized probabilistic discretization of x obtained in this way is always unique and can be formulated as

$$GPD(x) = \Big\{(I, p) \,\Big|\, I = f[I_1, ..., I_n], \\ p = \prod_{i=1}^{n} p_i;\ \forall\,(I_i, p_i) \in GPD(e_i); \\ i = 1, ..., n\Big\}$$

where notation $f[I_1, ..., I_n]$ is defined in Theorem 2.3 and has been proved to be an interval.

The convergence of $GPD(x)$ follows from Theorems 2.3 and 2.1. It can be summarized as

Theorem 3.3 *(Convergence of Primitive PDV Arithmetic)*
For all PDVs that are derived from primitive PDVs via algebraic expressions that are continuous on the Cartesian products of the closures of the bins of primitive PDVs, the generalized probabilistic discretizations calculated from Primitive PDV Arithmetic converge to the PDVs as the widths of the generalized probabilistic discretizations of all primitive PDVs tend to zero.

Definition 3.2 *(Exact Generalized Probabilistic Discretization)*
Let $\{e_1, ..., e_n\}$ be the pre-image primitive PDV set of PDV x, and $x = f(e_1, ..., e_n)$ where f is a continuous algebraic expression on the Cartesian products of the bins of $e_1, ..., e_n$. The exact generalized probabilistic discretization *of x with respect to $GPD(e_1)$, ..., $GPD(e_n)$, denoted by $EGPD(x)$, is defined by*

$$EGPD(x) = \left\{ (I, p) \,\middle|\, I = f(I_1, ..., I_n), \; p = \prod_{i=1}^{n} p_i; \; \forall (I_i, p_i) \in GPD(e_i); \; i = 1, ..., n \right\}$$

where $f(I_1, ..., I_n)$ is the function image of f over $I_1 \times \cdots \times I_n$.

Theorem 3.4 *(Continuity of Exact Generalized Probabilistic Discretization)*
Let $e_1, ..., e_n$ be independent PDVs with generalized probabilistic discretizations $GPD(e_i)$ and $GPD_k(e_i)$ where $i = 1, ..., n$ and $k = 1, 2, \cdots$. Let $x = f(e_1, ..., e_n)$ where f is a continuous algebraic expression on the Cartesian products of the bins of $e_1, ..., e_n$. Suppose that $GPD_k(e_i) \to GPD(e_i)$ as $k \to \infty$ for all i. Then

$$\lim_{k \to \infty} EGPD_k(x) = EGPD(x)$$

provided one of the following two conditions is satisfied:

(1) The bins in GPD(e_i) are finite intervals for all i.
(2) $GPD_k(e_i) \subseteq GPD(e_i)$ for all k and i.

Proof: See [31].

Definition 3.3 *(Exact Probability Distribution Bounds and Exact Probability Distribution Box)*
The p-bounds and p-box determined by the exact generalized probabilistic discretization of a PDV with respect to the generalized probabilistic discretizations of the variables in the pre-image primitive PDV set of the PDV are called the exact p-bounds and exact p-box of the PDV with respect to the generalized probabilistic discretizations of the variables in the pre-image primitive PDV set of the PDV, respectively.

We extend Primitive PDV Arithmetic as *Refined Primitive PDV Arithmetic* that can calculate the exact generalized probabilistic discretization of a PDV with respect to the generalized probabilistic discretizations of the primitive PDVs in the pre-image primitive PDV set.

Definition 3.4 *(Refined Primitive PDV Arithmetic)*
Let $\{e_1, ..., e_n\}$ be the pre-image primitive PDV set of PDV x, and $x = f(e_1, ..., e_n)$ where f is a continuous algebraic expression on the Cartesian products of the bins of $e_1, ..., e_n$. Refined Primitive PDV Arithmetic is a set of rules to determine a special generalized probabilistic discretization of x with respect to $GPD(e_1)$, ..., $GPD(e_n)$ by performing the following steps:

- *For $1 \le i \le n$, refine each bin of e_i to obtain a refined generalized probabilistic discretization of e_i, denoted by $RGPD(e_i)$, which has the representation*

$$RGPD(e_i) = \left\{ (I_{ik_i}, \, p_{ik_i}) \,\middle|\, k_i = 1, ..., l_i; \; \bigcup_{k_i=1}^{l_i} I_{ik_i} = I_i, \; \sum_{k_i=1}^{l_i} p_{ik_i} = p_i; \; \forall (I_i, p_i) \in GPD(e_i) \right\}.$$

- *Apply Primitive PDV Arithmetic on $RGPD(e_i)$, $1 \le i \le n$, to obtain a refined generalized probabilistic discretization of x, which is given by*

$$
\begin{aligned}
RGPD(x) = \Bigg\{ \Big(& f[I_{1k_1}, ..., I_{nk_n}] \, , \prod_{i=1}^{n} p_{ik_i} \Big) \\
& \Big| \; k_i = 1, ..., l_i; \; i = 1, ..., n; \\
& \bigcup_{k_i=1}^{l_i} I_{ik_i} = I_i, \; \sum_{k_i=1}^{l_i} p_{ik_i} = p_i; \\
& \forall \, (I_i, p_i) \in GPD(e_i) \Bigg\} \, .
\end{aligned}
$$

- *Combine all the bins in $RGPD(x)$ that are obtained from the same bins in $GPD(e_i)$, $1 \le i \le n$, and obtain a derived set that is an interval when f is continuous from the proof of the following Theorem 3.5. Make this interval a bin in a new generalized probabilistic discretization, and calculate the probability associated with this bin by multiplying the probabilities on the corresponding bins in $GPD(e_i)$, $1 \le i \le n$.*
- *The new generalized probabilistic discretization obtained from above is a* special *generalized probabilistic discretization of x with respect to $GPD(e_1)$, ..., $GPD(e_n)$, denoted by $SGPD(x)$, and has the representation*

$$
\begin{aligned}
& SGPD(x) \\
= & \Bigg\{ \Big(\bigcup_{\substack{1 \le k_i \le l_i \\ 1 \le i \le n}} f[I_{1k_1}, ..., I_{nk_n}] \, , \prod_{i=1}^{n} p_i \Big) \\
& \Big| \; \bigcup_{k_i=1}^{l_i} I_{ik_i} = I_i, \; \sum_{k_i=1}^{l_i} p_{ik_i} = p_i; \\
& \forall \, (I_i, p_i) \in GPD(e_i); \; i = 1, ..., n \Bigg\} \, .
\end{aligned}
$$

Theorem 3.5 *(Significance and Convergence of Refined Primitive PDV Arithmetic)*
Let $\{e_1, ..., e_n\}$ be the pre-image primitive PDV set of PDV x, and $x = f(e_1, ..., e_n)$ where f is a continuous algebraic expression on the Cartesian products of the bins of $e_1, ..., e_n$. Then $SGPD(x)$ computed via Refined Primitive PDV Arithmetic is a generalized probabilistic discretization of x, and

$$
EGPD(x) \subseteq SGPD(x) \, . \tag{3.1}
$$

Furthermore,

(1) if each bin in $GPD(e_i)$ is a finite interval for $1 \le i \le n$, and f is continuous on the Cartesian products of the closures of the bins in $GPD(e_1), ..., GPD(e_n)$, then

$$
\begin{aligned}
& \lim_{\substack{\|RGPD(e_i)\| \to 0 \\ 1 \le i \le n}} SGPD(x) \\
& = EGPD(x) \, ;
\end{aligned} \tag{3.2}
$$

(2) if there are sequences $GPD_k(e_i)$ ($k = 1, 2, ...$) such that each bin in $GPD_k(e_i)$ is a finite interval, f is continuous on the Cartesian products of the closures of the bins in $GPD_k(e_1), ..., GPD_k(e_n)$, $GPD_k(e_i) \subseteq GPD(e_i)$ and $\lim_{k \to \infty} GPD_k(e_i) = GPD(e_i)$ for each i, $1 \le i \le n$, then

$$
\begin{aligned}
& \lim_{k \to \infty} \lim_{\substack{\|RGPD_k(e_i)\| \to 0 \\ 1 \le i \le n}} SGPD_k(x) \\
& = \lim_{k \to \infty} EGPD_k(x) \\
& = EGPD(x) \, .
\end{aligned} \tag{3.3}
$$

Proof: See [31].

Remark: Inclusion relation (3.1) implies that the bounds given by $SGPD(x)$ enclose all the possible distributions of x. When all input bins are finite intervals, Relation (3.2) guarantees $SGPD(x)$ converges to $EGPD(x)$ that determines exactly the set of all the possible distributions of x, as the input bins are refined. When there is an input bin with infinite length or with singularity at one or two of its endpoints, Relation (3.3) guarantees that proper finite truncations on the infinite-length input bins or to exclude the singularities can be performed to achieve convergent approximations.

Combining Theorems 3.4 and 3.5 we have

Corollary 3.6 *(Stability of Refined Primitive PDV Arithmetic)*
In Refined Primitive PDV Arithmetic, small perturbations of the input bins do not lead to big changes in the results provided that all input bins are finite intervals.

Finally, we have

Definition 3.5 *(PDV Arithmetic)*
Primitive PDV Arithmetic and Refined Primitive PDV Arithmetic together are called PDV Arithmetic.

4 Implementation of PDV Arithmetic

PDV arithmetic is implemented in computer languages by including PDV as a basic data type. The implementation includes three aspects: 1. PDV recording; 2. preprocessor; and 3. subroutine library.

4.1 PDV Recording

As a basic data type, all PDV names in a program must be declared in the declarative statement with the same priority as other data types such as *integer* and *real.* These PDVs are called *declared PDVs.* There is another kind of PDV that appears when intermediate temporary variables are needed in the event that an algebraic expression is decomposed into a sequence of unitary and binary operations. We call them *intermediate PDVs.* Declared PDVs and intermediate PDVs are derived PDVs, i.e. they are functions of primitive PDVs.

PDVs are recorded by natural numbers in the order that they appear in the program. There is a one-to-one correspondence between PDVs and a finite sequence of natural integers starting from 1. We call these integers the *labels* or *indices* of the corresponding PDVs. For example, if we define PDVs x, y, and z in a program and no other PDVs are defined before them, then x, y, and z are recorded as PDV 1, 2, and 3 in the PDV list. Similarly, all primitive PDVs are identified by their indices that are the orders in which they are input by the user. For example, primitive PDV 2 is the second PDV input.

The generalized probabilistic discretization of a PDV is characterized by its bins and the probabilities associated to the bins. Each bin has the left and right endpoints. Thus, in principle we can use three arrays (left endpoint array, right endpoint array and probability array) to store all the information about the generalized probabilistic discretization of a PDV. Similarly, left endpoint array, right endpoint array and probability array are also used for primitive PDV.

In PDV Arithmetic, refinement is the key method for computing arbitrarily tight upper and lower bounds for a bin in the exact generalized probabilistic discretization of a PDV. As stated in Theorem 3.5, the accuracy of the approximation to the exact bounds does not depend on the way how the refinement is chosen, but only depends on the width of the refinement. Thus in the implementation, uniform subdivision is a simple and natural refinement to be used in such a way that a bin divisor (denoted by BIN_DIVISOR in our PDVFOR77 implementation) that is a positive number uniformly subdivides each input bin and the associated probability of the bin to achieve refined generalized probabilistic discretizations of primitive PDVs. The larger the bin divisor, the better the approximation. However, if we let m be the bin divisor, n be the total number of primitive PDVs in the program, then the total number of bins under consideration in the program would be proportional to m^n. Thus the chosen value of bin divisor is limited by the storage capacity and speed of the computer.

Back to the PDV recording it is crucial to distinguish by usage those arrays that store the endpoints of bins and the associated probabilities. There are four types of these arrays, two of them for primitive PDV and the other two for derived PDV. Namely, for each primitive PDV there are three arrays used for original inputs and three other arrays for the refined inputs; for each derived PDV there are three arrays used for the generalized probabilistic discretizations derived from the refined inputs and three other arrays for the special generalized probabilistic discretizations derived from regrouping (described in Definition 3.4).

As we have known, every bin of a derived PDV is obtained from the bins of the variables in the pre-image primitive PDV set via interval arithmetic. Thus the index of a bin in the generalized probabilistic discretization of a derived PDV is actually a function of the indices of the bins of the variables in the pre-image primitive PDV set. To account for the relationships among PDVs, we need to establish a portfolio for each derived PDV. This portfolio includes:

(1) The pre-image primitive PDV set of the derived PDV;
(2) A quantity to reflect the relationship of a bin of the derived PDV and the corresponding bins of the variables in its pre-image primitive PDV set. This quantity is represented by the *order* (or *index*) in which the bins of the derived PDV are arranged.

We construct a *primitive index set* for each derived PDV. The primitive index set is a set of tuples out of the pre-image primitive PDV set of the derived PDV. The number of dimensions of the tuple is the number of primitive PDVs in the pre-image primitive PDV set of the derived PDV. Each dimension of the tuple corresponds to a variable in the pre-image primitive set, and the value of the dimension index ranges from 1 to the number of bins of this primitive PDV. Obviously, the primitive index set of a PDV records all the indices of the bins of the variables in its pre-image PDV set. To determine the index of a bin of a derived PDV from its primitive index set, the following relation (4.1) is used.

Assume that there is an n-D index set

$$\left\{(i_1, ..., i_n) \,\middle|\, 1 \le i_r \le m_r,\ 1 \le r \le n\right\}.$$

This index set can be mapped to a 1-D index set via

$$(i_1, ..., i_n) \longmapsto \sum_{j=1}^{n-1} (i_j - 1) \prod_{r=j+1}^{n} m_r + i_n\,. \tag{4.1}$$

Mapping (4.1) is a one-to-one mapping. Using (4.1) to arrange the order of the bins of the derived PDV implies that the index of a bin of the derived PDV uniquely corresponds to some indices of the bins of the variables in the pre-image primitive PDV set. Therefore, arranging the order of the bins of a derived PDV in this way completely reflects the information in the pre-image primitive PDV set.

An illustrative example is given as follows. Consider a PDV that is a function of two inputs: primitive PDVs 1 and 3. Primitive PDV 1 has 30 bins, and primitive PDV 3 has 40 bins. Then this PDV has $30 \times 40 =$

1200 bins. Choose an arbitrary bin, say, the 1000th bin of this PDV. Using formula (4.1), we obtain an integer equation:

$$40\,(i_1 - 1) + i_2 = 1000,$$
$$1 \leq i_1 \leq 30\,, \quad 1 \leq i_2 \leq 40\,.$$

The unique solution to the equation is $i_1 = 25$, $i_2 = 40$. This means that there is a one-to-one correspondence between the 1000th bin of this PDV and the pair of the 25th bin of primitive PDV 1 and the 40th bin of primitive PDV 3. This relationship is used to calculate the interval and the probability of the 1000th bin of this PDV as well as to manipulate the correlations between this bin and the bins of other PDVs.

Relation (4.1) also provides a formula to compute exact generalized probabilistic discretization. Recall that in Refined Primitive PDV Arithmetic, we subdivide each bin of primitive PDVs to get refined generalized probabilistic discretization. Each refined bin is given a label, called *exact-label*, that points to the original input bin which it comes from. As a result, each bin of a derived PDV that is derived from the refinement has an exact-label that is derived from the exact-labels of the bins of the variables in its pre-image primitive PDV set. If we assume that n is the cardinality of the pre-image primitive PDV set (in which the primitive PDVs are listed in ascending order by their indices) of a PDV, i_r is the exact-label of the refined bin of the primitive PDV that is located at position r in the pre-image primitive PDV set ($1 \leq r \leq n$), and m_r is the number of bins of this primitive PDV at position r before being refined, then the right hand side of (4.1) defines the exact-label of the derived bin of the derived PDV. All the bins of the derived PDV with the same exact-label come from the same bins of the original PDV inputs (that are not refined) in the pre-image primitive PDV set. Combining the bins of the derived PDV that have the same exact-label, we obtain a bin and its associated probability in the special generalized probabilistic discretization that converges to the exact generalized probabilistic discretization of the PDV as the bin divisor tends to ∞.

4.2 Preprocessor

The new data type PDV in a computer language needs to be parsed in order to be compatible with the existing computer compilers. We use a preprocessor for this task. The preprocessor can be written in the scripting language PERL because of the portability of PERL. After the parsing, a number of files are generated and they, along with the neccessary subroutine library, can be compiled by the compilers to generate an executable output file.

In the current implementation, PDV data type can be used only in main program. The primary functions of the preprocessor are in three aspects: 1. declarative statements; 2. input PDV data; and 3. executable statements. Besides, some basic simplifications are automatically done by default by the preprocessor to reduce the uncertainties of PDVs, unless the user explicitly deactivates this feature.

We will now describe the primary functions of the preprocessor in parsing.

(1) <u>Declarative statements</u>
All PDVs that appear in the source program must be declared as other existing data types, such as *real* and *integer*. Each PDV can be declared only once. The syntax for declaring PDVs is as follows.

pdv variable 1, ..., variable n

All the PDV variable names are placed in a symbol table and labeled in the order as they appear. These labels are stored in a PDV label list that are used as pointers to the PDVs in the program. The intermediate PDVs generated by parsing algebraic expressions are also recorded in the PDV label list at the time they appear. The preprocessor should be able to reuse these temporary variables in the expressions to minimize the total number of temporary variables in the list.

(2) Input PDV data

When data are input to a PDV by the user, a new primitive PDV is enumerated such that the PDV and the corresponding primitive PDV share the same data.

The syntax for inputting PDV data is as follows. Assume x is a PDV, then the statements

$$\mathrm{bin}[x] = [a_1, b_1; a_2, b_2; ...; a_n, b_n]$$

$$\mathrm{df}[x] = [p_1, p_2, ..., p_n]$$

$$\mathrm{cdf}[x] = [q_1, q_2, ..., q_n]$$

define x with n bins $[a_1, b_1]$, $[a_2, b_2]$, ..., $[a_n, b_n]$, with probabilities p_1, p_2, ..., p_n on each bin, and with cumulative probabilities q_1, q_2, ..., q_n up to each bin, respectively. a_i, b_i, p_i, q_i $(1 \leq i \leq n)$ can be numbers, array elements, or algebraic expressions of non-PDV variables. When they are numbers, requirements such as $a_i \leq b_i$, $p_i \geq 0$ for $1 \leq i \leq n$, and $0 \leq q_1 \leq \cdots \leq q_n$ must be satisfied, otherwise the preprocessor will print out the error messages.

There is a restriction on the orders of the PDV input statements. *The* bin *input statement must appear before the* df *and* cdf *input statements for the same input PDV.* Besides, the *bin* statement can appear by itself without the *df* and *cdf* statements. When there is the *bin* statement only, the preprocessor assumes that each bin gets its probability from the portion of its width out of the total widths of all the bins, that is, it calculates the corresponding probabilities on the bins based on the formula

$$p_i = \frac{b_i - a_i}{\sum_{r=1}^{n}(b_r - a_r)}\ , \qquad 1 \leq i \leq n\,.$$

If there are *df* or *cdf* statements for the same PDV name in the following part of the program, the preprocessor will update the data in the probability arrays while keeping the information about the bins unchanged. If a new *bin* statement is defined for the same PDV name, all the previous information about this derived PDV is overridden and a new primitive PDV is defined.

The *cdf* statement is fragile in the sense that it requires the already-existing bin statement to have ascending endpoints, i.e., $a_1 \leq b_1 \leq a_2 \leq b_2 \cdots \leq a_n \leq b_n$. The preprocessor parses the *cdf* statement by calculating the corresponding probability for each bin using subtraction. Notice that PDV Arithmetic needs only associated probabilities on the bins in the generalized probabilistic discretization. Hence *cdf* statement is designed only for the user to have the convenience to input cumulative probabilities in the program.

There is some flexibility for the user when inputting probabilities. The values in *df* and *cdf* statements can be treated as weights, where the values in *df* statement need not sum to 1, and the last value in *cdf* statement need not be 1. All values in *df* and *cdf* statements are required to be nonnegative, and the values in *cdf* statement must be increasing. The preprocessor can normalize the input weights automatically.

(3) Executable statements
The executable statements includes assignment statements and the output statements. The PERL preprocessor parses them into a sequence of calls to the subroutines in the subroutine library.

4.3 Subroutine Library

The subroutine library designed for PDV Arithmetic is no doubt the most important part of the implementation. It provides many subroutines to answer the subroutine calls resulting from parsing the statements that contain PDV data type.

Some often-used probability distribution functions can be stored in the subroutine library. Proper truncations are needed for infinite interval domain in order to receive finite probabilistic discretization. For example, the standard normal distribution function can be stored in the following way. We truncate the tails $(-\infty, -5)$ and $(5, \infty)$ by losing probability $0.5 \cdot 6.981 \cdot 10^{-7}$ on each tail. We then divide interval $[-5, 5]$ into 1000 bins, and get the cumulative probability for each breaking point from any mathematical or statistical handbook (e.g., see [28]). To make up the probabilities lost from the truncation, distribution function values 0 and 1 are assigned to the breaking point -5 and 5, respectively. Because any normal distribution is derived from the combination of its mean, its standard derivation, and the standard normal distribution, a probabilistic discretization of any normal distribution with 1000 equal-width bins can be generated from the above stored standard normal distribution. In practice, the user can use a global parameter BIN_NUM_NORMAL to control the actual number of bins of the normal distribution. A subroutine in the library can generate a probabilistic discretization for the desired normal distribution from the stored standard normal distribution with the bin number specified by the user via this parameter.

5 Examples

As applications of PDV Arithmetic, some examples are presented. The computations in the examples are programmed in PDVFOR77, which is an extension of Fortran 77 by adding PDV as a basic data type. The reader can download PDVFOR77 from `http://math.lanl.gov/~liw` [30].

Example 5.1 *Consider a* 2×2 *matrix* $\mathbf{A} = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$, *where the parameters* a, b, c, d *are PDVs. The parameters can be linearly dependent, nonlinearly dependent, partially dependent, and independent. The following 4 cases will demonstrate the comparisons for the density functions of the eigenvalues of this matrix between PDV Arithmetic and Monte Carlo simulation providing the different relationships among* a, b, c, d. *The eigenvalues of the matrix are the roots of the following characteristic equation*

$$(\lambda - a)(\lambda - d) - bc = 0\,.$$

(1) a, b, c, d *are linearly dependent PDVs, where* $a = 0.5d + 1.5$, $c = 2a - 1$, $b = 0.5c - 0.5$, $d \in [-1, 1]$ *and* d *is uniformly distributed. The comparisons are shown in Figure 5.1.*

(2) a, b, c, d *are nonlinearly dependent PDVs, where* $a = 3d^2 + 1$, $c = 2a - 1$, $b =$

$0.5c/a + 0.5$, $d \in [-1, 1]$ *and* d *is uniformly distributed. The comparisons are shown in Figure 5.2.*

(3) a, b, c, d *are partially dependent PDVs, where* $a = 3d^2 + 1$, $b = 0.5c/a + 0.5$, $c \in [2, 10]$, $d \in [-1, 1]$, c *and* d *are uniformly distributed and independent. The comparisons are shown in Figure 5.3.*

(4) a, b, c, d *are independent PDVs, where* $a \in [1, 2]$, $b \in [0, 1]$, $c \in [1, 3]$, $d \in [-1, 1]$, *and all are uniformly distributed. The comparisons are shown in Figure 5.4.*

The next example is from the challenge problem set given in Oberkampf et al. [38].

Example 5.2 *(Challenge Problem Set)*
Let the model of the physical process is given by

$$y = (a + b)^a\,. \tag{5.1}$$

The parameters a and b are independent real numbers, i.e., knowledge about the value of one parameter implies nothing about the value of the other, and $a \geq 0$, $b > 0$. The task for each problem in the sequence is to quantify the uncertainty in y given the information regarding a and b. In other words, what can be ascertained about the response of the system y, given only the stated information about a and b?

The sequence of problems begins with very little information concerning a and b so that they are only known to lie within specified intervals. Information of different types is incrementally added in each subsequent problem in the sequence by way of more specificity concerning the parameters. The information given may be mutually supportive, or some of it may be contradictory to some degree.

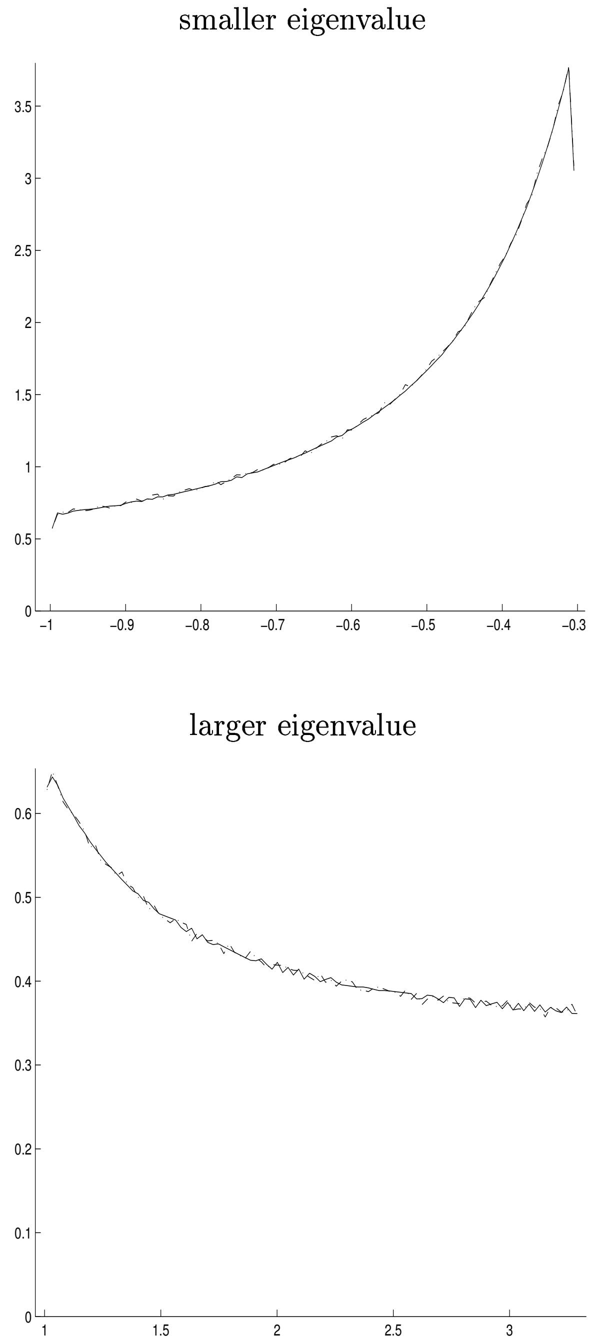

Fig. 5.1. Comparisons for the density functions of the eigenvalues between PDV Arithmetic (solid line) and Monte Carlo simulation (dot-dashing line) for case (1) in Example 5.1, where all parameters are linearly dependent.

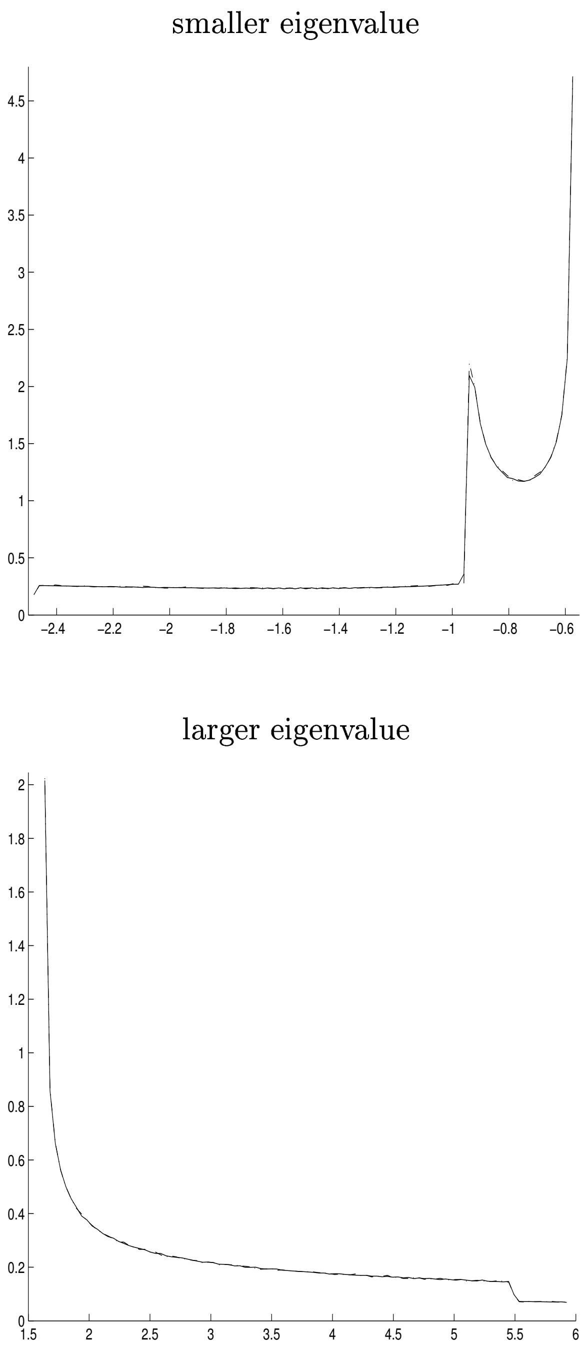

Fig. 5.2. Comparisons for the density functions of the eigenvalues between PDV Arithmetic (solid line) and Monte Carlo simulation (dot-dashing line) for case (2) in Example 5.1, where all parameters are nonlinearly dependent.

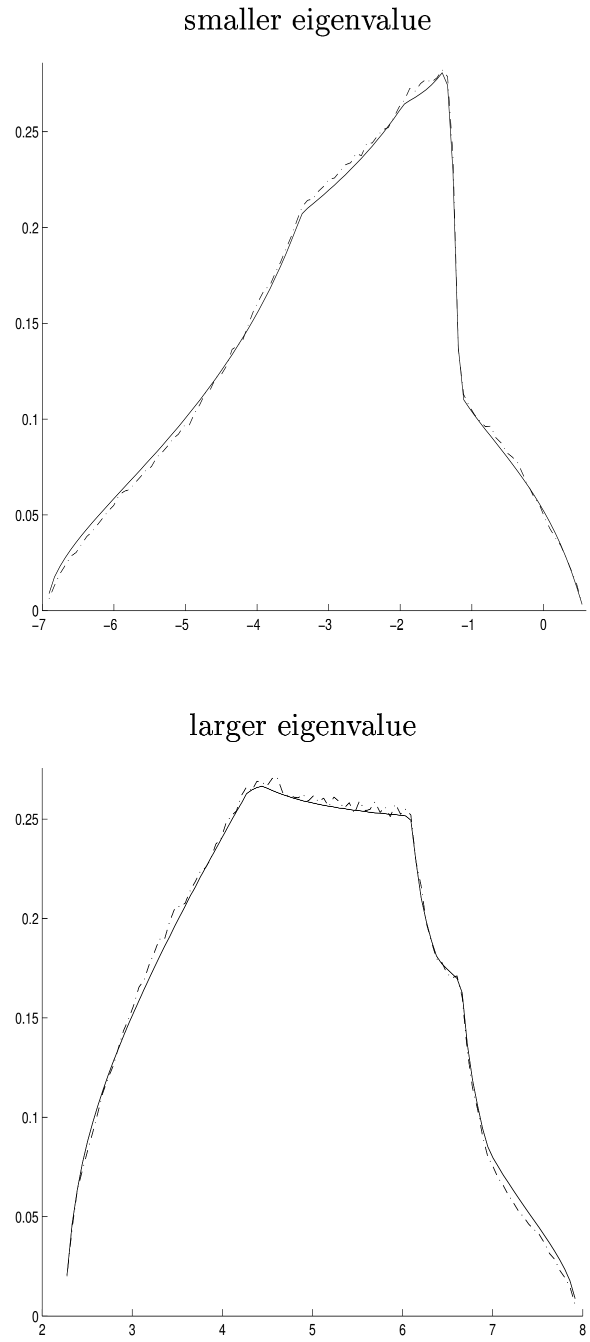

Fig. 5.3. Comparisons for the density functions of the eigenvalues between PDV Arithmetic (solid line) and Monte Carlo simulation (dot-dashing line) for case (3) in Example 5.1, where all parameters are partially dependent.

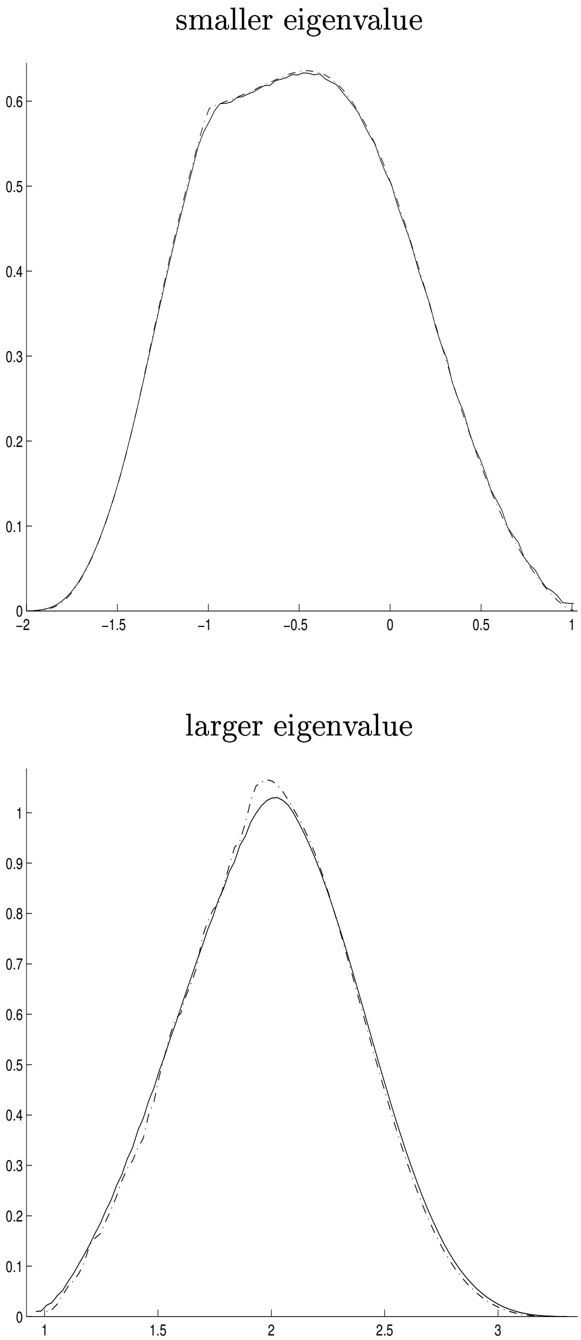

Fig. 5.4. Comparisons for the density functions of the eigenvalues between PDV Arithmetic (solid line) and Monte Carlo simulation (dot-dashing line) for case (4) in Example 5.1, where all parameters are independent.

Six problems are specified in the sequence as follows.

Problem 1: a and b are in an interval, respectively.

Problem 2: a is in an interval, b is characterized by multiple intervals.

Problem 3: a and b are characterized by multiple intervals, respectively.

Problem 4: a is in an interval, b is specified by a probability distribution with imprecise parameters.

Problem 5: a is characterized by multiple intervals, b is specified by a probability distribution with imprecise parameters.

Problem 6: a is in an interval, b is specified by a precise probability distribution.

The solutions to Problems 1, 2, and 3 can be obtained analytically by finding the global maximal and minimal values for function (5.1) over some interval domains. However, the rest of the problems requires the consideration of precise or imprecise probability distributions. In this example, we consider using upper and lower p-bounds is an appropriate way to quantify the uncertainties in the above problem set. PDV Arithmetic provides such a computational approach.

Along with the description of each problem in the sequence is an illustration in which we input numerical values for the parameters in the problem set. Accordingly, parameters a and b are predicted in some independent intervals with equal probabilities. We then obtain generalized probabilistic discretizations for each of them. We plot the p-box that approximates the exact p-box of output y with respect to each of these input generalized probabilistic discretizations for a and b. To make a contrast, we plot the refined p-box of y with respect to the refined probabilistic discretizations of a and b that are obtained by evenly dividing each input bin using a sufficiently large bin divisor. The refined p-box gives the shape and range of the limiting probability distribution function that is determined by additional assumption that each parameter is evenly distributed on each input bin.

Problem 1: a and b are contained in closed intervals $A = [a_1, a_2]$ and $B = [b_1, b_2]$, respectively.

Illustration 5.2.1: *Numerical values for parameters in Problem 1 are:*

$$A = [0.1, 1.0]\,, \quad B = [0.0, 1.0]\,.$$

We choose BIN_DIVISOR $= 400$. *The p-box and refined p-box are plotted in Figure 5.5. Note that the analytic solution to the range interval of y is* $[(e^{-1})^{e^{-1}}, 2.0] \approx [0.6922, 2.0]$.

Problem 2: a is contained in closed interval $A = [a_1, a_2]$, and b is given by n independent and equally credible closed intervals $B_j = [b_1^j, b_2^j]$, where $j = 1, ..., n$. Given this information, consider the following family of problems.

2a) B_j is a consonant collection of intervals (i.e., nested intervals). Without loss of generality, we assume $B_j \subset B_{j+1}$, for $j = 1, ..., n-1$.

2b) B_j is a consistent collections of intervals (i.e., having non-empty overall intersection). Without loss of generality, we assume $\bigcap_{j=1}^{n} B_j \neq \phi$.

2c) B_j is an arbitrary collection of intervals.

Illustration 5.2.2a: *Numerical values for parameters in Problem 2a are:*

$$\begin{aligned} &n = 4\,; \\ &A = [0.1, 1.0]\,; \\ &B_1 = [0.6, 0.8]\,, \quad B_2 = [0.4, 0.85]\,, \\ &B_3 = [0.2, 0.9]\,, \quad B_4 = [0.0, 1.0]\,. \end{aligned}$$

We choose BIN_DIVISOR $= 200$. *By the assumption that each source of information concerning b is equally credible, we assign equal probability* 0.25 *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.6.*

Illustration 5.2.2b: *Numerical values for parameters in Problem 2b are:*

$$\begin{aligned} &n = 4\,; \\ &A = [0.1, 1.0]\,; \\ &B_1 = [0.6, 0.9]\,, \quad B_2 = [0.4, 0.8]\,, \\ &B_3 = [0.1, 0.7]\,, \quad B_4 = [0.0, 1.0]\,. \end{aligned}$$

We choose BIN_DIVISOR $= 200$. *By the assumption that each source of information concerning b is equally credible, we assign equal probability* 0.25 *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.7.*

Illustration 5.2.2c: *Numerical values for parameters in Problem 2c are:*

$$\begin{aligned} &n = 4\,; \\ &A = [0.1, 1.0]\,; \\ &B_1 = [0.6, 0.8]\,, \quad B_2 = [0.5, 0.7]\,, \\ &B_3 = [0.1, 0.4]\,, \quad B_4 = [0.0, 1.0]\,. \end{aligned}$$

We choose BIN_DIVISOR $= 200$. *By the assumption that each source of information concerning b is equally credible, we assign equal probability* 0.25 *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.8.*

Problem 3: a is given by m independent and equally credible intervals $A_i = [a_1^i, a_2^i]$, where $i = 1, ...m$, and b is given by n independent and equally credible closed intervals $B_j = [b_1^j, b_2^j]$, where $j = 1, ..., n$.

Given this information, consider the following family of problems.

3a) A_i and B_j are consonant collections of intervals (i.e., nested intervals). Without loss of generality, we assume $A_i \subset A_{i+1}$, $i = 1, ..., m-1$; and $B_j \subset B_{j+1}$, for $j = 1, ..., n-1$.

3b) A_i and B_j are consistent collections of intervals (i.e., having non-empty overall intersection). Without loss of generality, we assume $\bigcap_{i=1}^{m} A_i \neq \phi$, and $\bigcap_{j=1}^{n} B_j \neq \phi$.

3c) A_i and B_j are arbitrary collections of intervals.

Illustration 5.2.3a: *Numerical values for parameters in Problem 3a are:*

$$\begin{aligned}
&m = 3\,, \quad n = 4\,;\\
&A_1 = [0.5, 0.7]\,, \quad A_2 = [0.3, 0.8]\,,\\
&A_3 = [0.1, 1.0]\,;\\
&B_1 = [0.6, 0.6]\,, \quad B_2 = [0.4, 0.85]\,,\\
&B_3 = [0.2, 0.9]\,, \quad B_4 = [0.0, 1.0]\,.
\end{aligned}$$

We choose BIN_DIVISOR = 120*. By the assumption that each source of information concerning a and b is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a, and equal probability* 0.25 *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.9.*

Illustration 5.2.3b: *Numerical values for parameters in Problem 3b are:*

$$\begin{aligned}
&m = 3\,, \quad n = 4\,;\\
&A_1 = [0.5, 1.0]\,, \quad A_2 = [0.2, 0.7]\,,\\
&A_3 = [0.1, 0.6]\,;\\
&B_1 = [0.6, 0.6]\,, \quad B_2 = [0.4, 0.8]\,,\\
&B_3 = [0.1, 0.7]\,, \quad B_4 = [0.0, 1.0]\,.
\end{aligned}$$

We choose BIN_DIVISOR = 120*. By the assumption that each source of information concerning a and b is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a, and equal probability* $1/4$ *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.10.*

Illustration 5.2.3c: *Numerical values for parameters in Problem 3c are:*

$$\begin{aligned}
&m = 3\,, \quad n = 4\,;\\
&A_1 = [0.8, 1.0]\,, \quad A_2 = [0.5, 0.7]\,,\\
&A_3 = [0.1, 0.4]\,;\\
&B_1 = [0.8, 1.0]\,, \quad B_2 = [0.5, 0.7]\,,\\
&B_3 = [0.1, 0.4]\,, \quad B_4 = [0.0, 0.2]\,.
\end{aligned}$$

We choose BIN_DIVISOR = 120*. By the assumption that each source of information concerning a and b is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a, and equal probability* 0.25 *to each input bin of b in the source program. The p-box and refined p-box are plotted in Figure 5.11.*

Problem 4: a is contained in the closed interval A, and b is given by a log-normal probability distribution. One has

$$A = [a_1, a_2], \text{ and } \ln b \sim N(\mu, \sigma)$$

The value of the mean, μ, and the standard deviation, σ, are given, respectively, by the closed intervals

$$M = [\mu_1, \mu_2] \quad \text{and} \quad S = [\sigma_1, \sigma_2]\,.$$

Illustration 5.2.4: *Numerical values for parameters in Problem 4 are:*

$$A = [0.1, 1.0]\,,$$
$$M = [0.0, 1.0]\,, \quad S = [0.1, 0.5]\,.$$

We choose BIN_DIVISOR $= 20$. *We also choose* BIN_NUM_STD_NORMAL $= 20$*, i.e., the number of bins for the standard normal distribution in the range* $[-5, 5]$ *is* 20. *The p-box and refined p-box are plotted in Figure 5.12.*

Problem 5: The information concerning a is given by m independent sources of information. Each source specifies a closed interval A_i that contains the values for a. The information concerning b is given by n independent sources of information. Each source specifies b is given by a log-normal probability distribution, where the mean and standard deviation assume values in closed intervals, M_j and S_j, respectively. One has

$$A_i = [a_1^i, a_2^i]\,;$$
$$\ln b \sim N(\mu^j, \sigma^j)\,;$$
$$\mu^j \in M_j = [\mu_1^j, \mu_2^j]\,, \quad \sigma^j \in S = [\sigma_1^j, \sigma_2^j]\,;$$
$$i = 1, 2, ..., m\,; \quad j = 1, 2, ..., n\,.$$

The m sources of information for a are equally credible. So are the n sources for μ and σ. Thus we may consider accepting each source of information with equal probability for each parameter a, μ and σ, respectively. Given this information, consider the following family of problems:

5a) A_i, M_j, S_j are consonant collections of intervals (i.e., nested intervals), respectively. That is, $A_i \subset A_{i+1}$, for $i = 1, ..., m-1$; $M_j \subset M_{j+1}$, $S_j \subset S_{j+1}$, for $j = 1, ..., n-1$.

5b) A_i, M_j, S_j are consistent collections of intervals (i.e., having non-empty overall intersection), respectively. That is, $\bigcap_{i=1}^{m} A_i \neq \phi$, $\bigcap_{j=1}^{n} M_j \neq \phi$, $\bigcap_{j=1}^{n} S_j \neq \phi$.

5c) A_i, M_j, S_j are arbitrary collections of intervals.

Illustration 5.2.5a: *Numerical values for parameters in Problem 5a are:*

$$m = 3\,, \quad n = 3\,;$$
$$A_1 = [0.5, 0.7]\,, \quad A_2 = [0.3, 0.8]\,,$$
$$A_3 = [0.1, 1.0]\,;$$
$$M_1 = [0.6, 0.8]\,, \quad M_2 = [0.2, 0.9]\,,$$
$$M_3 = [0.0, 1.0]\,;$$
$$S_1 = [0.3, 0.4]\,, \quad S_2 = [0.2, 0.45]\,,$$
$$S_3 = [0.1, 0.5]\,.$$

We choose BIN_DIVISOR $= 8$*, and* BIN_NUM_NORMAL $= 20$. *By the assumption that each source of information concerning a,* μ *and* σ *is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a,* μ *and* σ *in the source program. The p-box and refined p-box are plotted in Figure 5.13.*

Illustration 5.2.5b: *Numerical values for parameters in Problem 5b are:*

$$m = 3\,, \quad n = 3\,;$$
$$A_1 = [0.5, 1.0]\,, \quad A_2 = [0.2, 0.7]\,,$$
$$A_3 = [0.1, 0.6]\,;$$
$$M_1 = [0.6, 0.9]\,, \quad M_2 = [0.1, 0.7]\,,$$
$$M_3 = [0.0, 1.0]\,;$$
$$S_1 = [0.3, 0.45]\,, \quad S_2 = [0.15, 0.35]\,,$$
$$S_3 = [0.1, 0.5]\,.$$

We choose BIN_DIVISOR $= 8$*, and* BIN_NUM_NORMAL $= 20$. *By the assumption that each source of information concerning a,* μ *and* σ *is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a,* μ *and* σ *in the source program. The p-box and refined p-box are plotted in Figure 5.14.*

Illustration 5.2.5c: *Numerical values for parameters in Problem 5c are:*

$$\begin{aligned} &m = 3\,, \quad n = 3\,; \\ &A_1 = [0.8, 1.0]\,, \quad A_2 = [0.5, 0.7]\,, \\ &A_3 = [0.1, 0.4]\,; \\ &M_1 = [0.6, 0.8]\,, \quad M_2 = [0.1, 0.4]\,, \\ &M_3 = [0.0, 1.0]\,; \\ &S_1 = [0.4, 0.5]\,, \quad S_2 = [0.25, 0.35]\,, \\ &S_3 = [0.1, 0.2]\,. \end{aligned}$$

We choose BIN_DIVISOR $= 8$*, and* BIN_NUM_NORMAL $= 20$*. By the assumption that each source of information concerning a, μ and σ is equally credible, respectively, we assign equal probability* $1/3$ *to each input bin of a, μ and σ in the source program. The p-box and refined p-box are plotted in Figure 5.15.*

Problem 6: a is contained in closed interval A, and b is given by a lognormal probability distribution. That is,

$$A = [a_1, a_2] \quad \text{and} \quad \ln b \sim N(\mu, \sigma)\,.$$

The values of μ and σ are precisely known.

Illustration 5.2.6: *Numerical values for parameters in Problem 6 are:*

$$A = [0.1, 1.0]\,, \quad \mu = 0.5\,, \quad \sigma = 0.5\,.$$

We choose BIN_DIVISOR $= 200$*, and* BIN_NUM_NORMAL $= 250$*. The p-box and refined p-box are plotted in Figure 5.16.*

Figures 5.5–5.16 are very suggestive. Problems 1 to 3 are arranged in a *monotone increasing order in information* about parameters a and b, that is, more and more information for parameters a and b are obtained. It is true that the more information, the more specific the generalized probabilistic discretization, thus the p-boxes are in a *nested order*, i.e., the latter p-box is contained in the former one. The figures illustrate how the p-boxes are in nested order. Problems 4 and 5, Problems 4 and 6 are

Illustration 5.2.1 for Problem 1

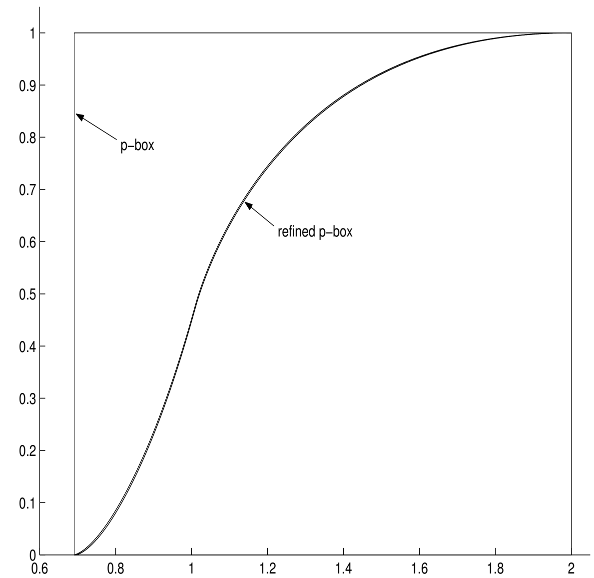

Fig. 5.5. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and b belongs to $[0.0, 1.0]$. We choose BIN_DIVISOR $= 400$.

Illustration 5.2.2a for Problem 2a

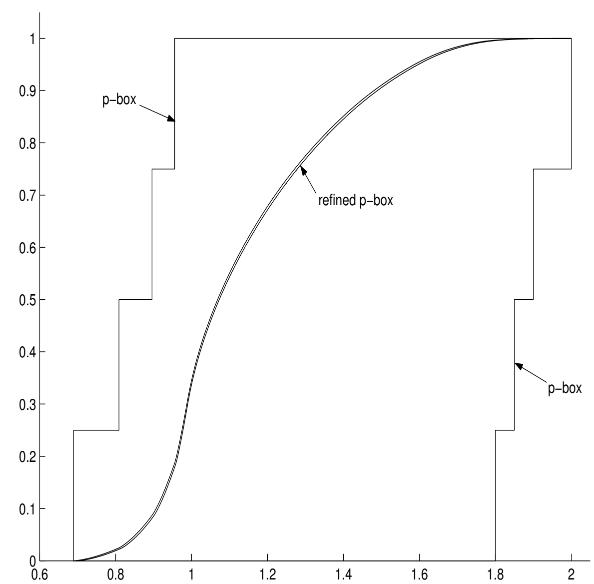

Fig. 5.6. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and b belongs to 4 independent nested intervals $[0.6, 0.8]$, $[0.4, 0.85]$, $[0.2, 0.9]$, and $[0.0, 1.0]$. We choose BIN_DIVISOR $= 200$.

Illustration 5.2.2b for Problem 2b

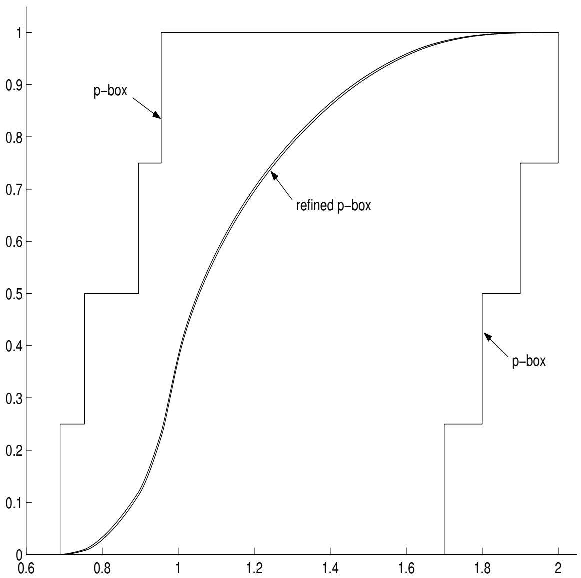

Fig. 5.7. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and b belongs to 4 independent consistent intervals $[0.6, 0.9]$, $[0.4, 0.8]$, $[0.1, 0.7]$, and $[0.0, 1.0]$. We choose BIN_DIVISOR = 200.

Illustration 5.2.2c for Problem 2c

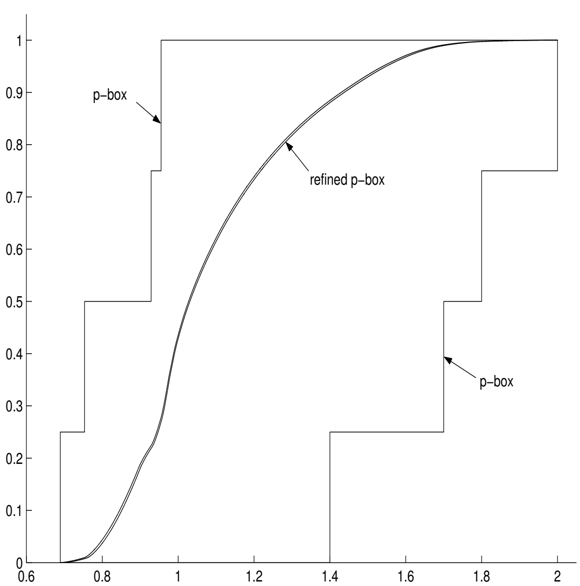

Fig. 5.8. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and b belongs to 4 independent intervals $[0.6, 0.8]$, $[0.5, 0.7]$, $[0.1, 0.4]$, and $[0.0, 1.0]$. We choose BIN_DIVISOR = 200.

Illustration 5.2.3a for Problem 3a

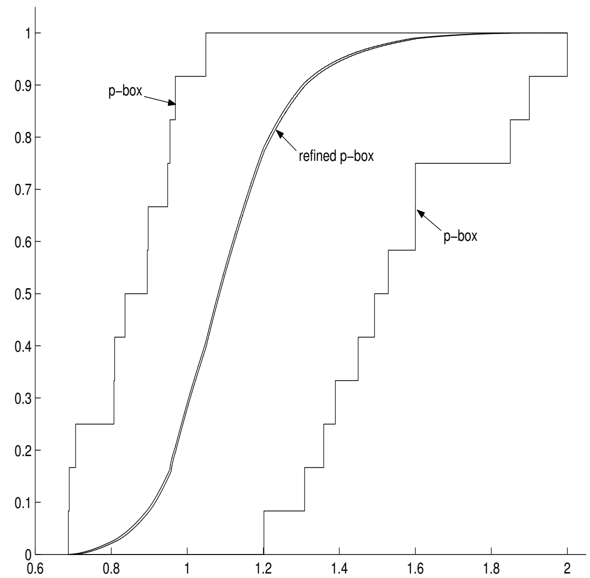

Fig. 5.9. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to 3 independent nested intervals $[0.5, 0.7]$, $[0.3, 0.8]$, $[0.1, 1.0]$, and b belongs to 4 independent nested intervals $[0.6, 0.6]$, $[0.4, 0.85]$, $[0.2, 0.9]$, $[0.0, 1.0]$. We choose BIN_DIVISOR = 120.

Illustration 5.2.3b for Problem 3b

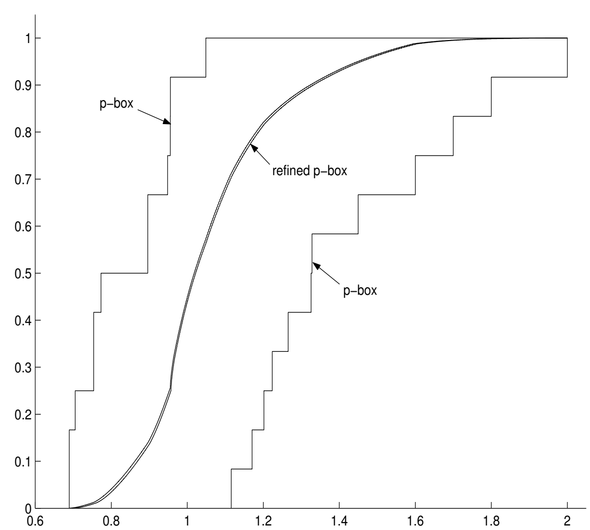

Fig. 5.10. The p-box and refined p-box for $y = (a + b)^a$, where a belongs to 3 independent consistent intervals $[0.5, 1.0]$, $[0.2, 0.7]$, $[0.1, 0.6]$, and b belongs to 4 independent consistent intervals $[0.6, 0.6]$, $[0.4, 0.8]$, $[0.1, 0.7]$, $[0.0, 1.0]$. We choose BIN_DIVISOR = 120.

Illustration 5.2.3c for Problem 3c

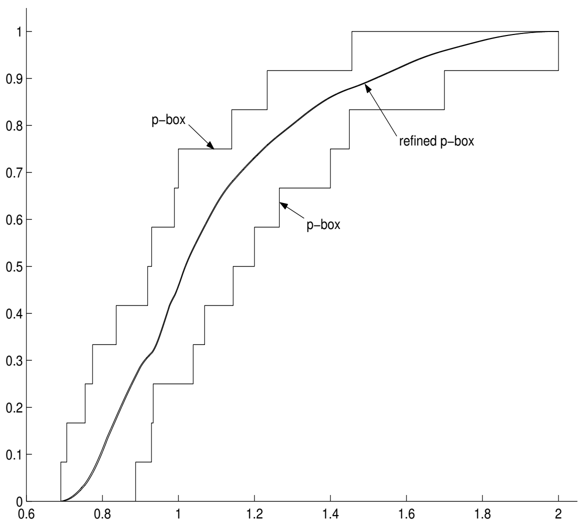

Fig. 5.11. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to 3 independent intervals $[0.8, 1.0]$, $[0.5, 0.7]$, $[0.1, 0.4]$, and b belongs to 4 independent intervals $[0.8, 1.0]$, $[0.5, 0.7]$, $[0.1, 0.4]$, $[0.0, 0.2]$. We choose BIN_DIVISOR = 120.

Illustration 5.2.4 for Problem 4

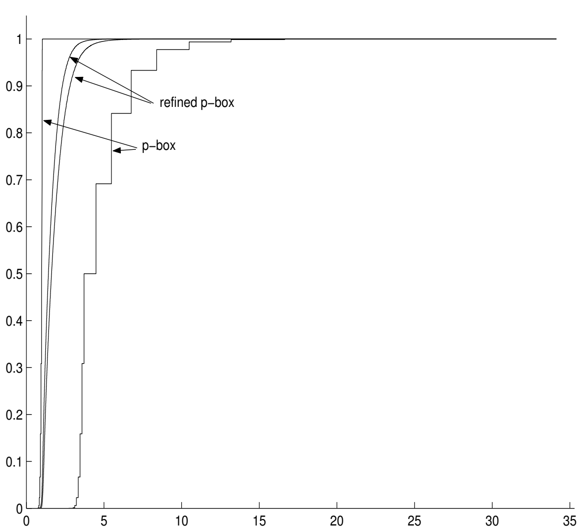

Fig. 5.12. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and $\ln b$ follows a normal distribution $N(\mu, \sigma)$ in which μ belongs to $[0.0, 1.0]$ and σ belongs to $[0.1, 0.5]$. We choose BIN_DIVISOR = 20, BIN_NUM_NORMAL = 20.

Illustration 5.2.5a for Problem 5a

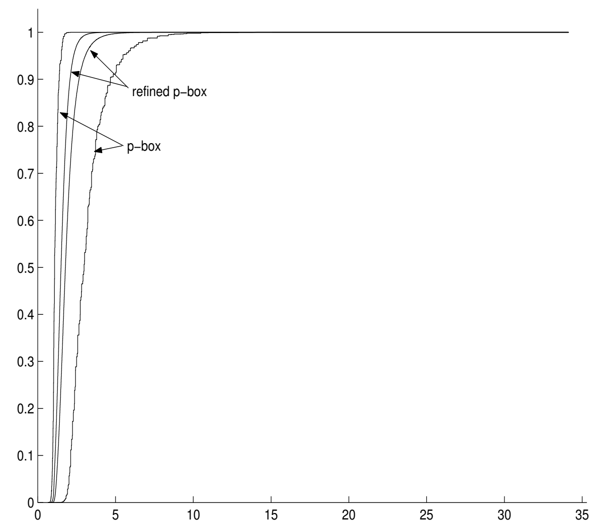

Fig. 5.13. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to 3 independent nested intervals $[0.5, 0.7]$, $[0.3, 0.8]$, $[0.1, 1.0]$, and $\ln b$ follows a normal distribution $N(\mu, \sigma)$ where μ belongs to 3 independent nested intervals $[0.6, 0.8]$, $[0.2, 0.9]$, $[0.0, 1.0]$, and σ belongs to 3 independent nested intervals $[0.3, 0.4]$, $[0.2, 0.45]$, $[0.1, 0.5]$. We choose BIN_DIVISOR = 8, BIN_NUM_NORMAL = 20.

Illustration 5.2.5b for Problem 5b

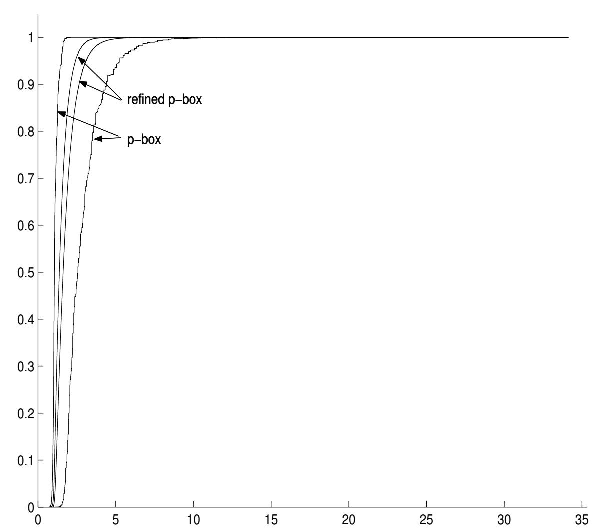

Fig. 5.14. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to 3 independent consistent intervals $[0.5, 1.0]$, $[0.2, 0.7]$, $[0.1, 0.6]$, and $\ln b$ follows $N(\mu, \sigma)$ where μ belongs to 3 independent consistent intervals $[0.6, 0.9]$, $[0.1, 0.7]$, $[0.0, 1.0]$, and σ belongs to 3 independent consistent intervals $[0.3, 0.45]$, $[0.15, 0.35]$, $[0.1, 0.5]$. We choose BIN_DIVISOR = 8, BIN_NUM_NORMAL = 20.

Illustration 5.2.5c for Problem 5c

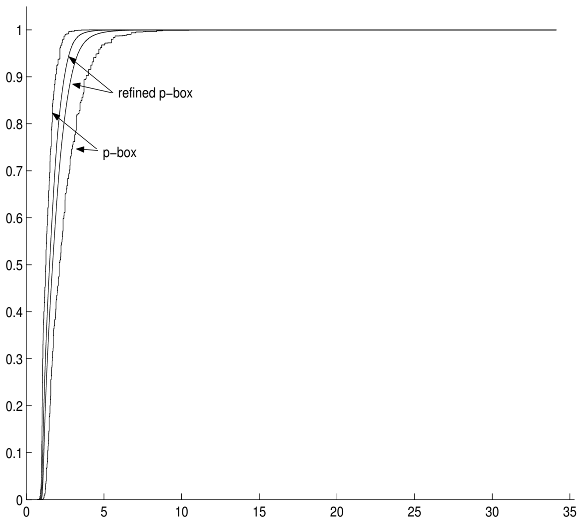

Fig. 5.15. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to 3 independent intervals $[0.8, 1.0]$, $[0.5, 0.7]$, $[0.1, 0.4]$, and $\ln b$ follows $N(\mu, \sigma)$ where μ belongs to 3 independent intervals $[0.6, 0.8]$, $[0.1, 0.4]$, $[0.0, 1.0]$, and σ belongs to 3 independent intervals $[0.4, 0.5]$, $[0.25, 0.35]$, $[0.1, 0.2]$. We choose BIN_DIVISOR = 8, BIN_NUM_NORMAL = 20.

Illustration 5.2.6 for Problem 6

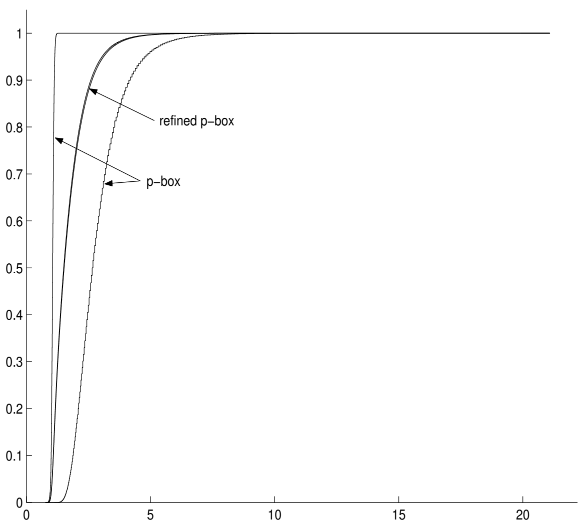

Fig. 5.16. The p-box and refined p-box for $y = (a+b)^a$, where a belongs to $[0.1, 1.0]$, and $\ln b$ follows the normal distribution $N(0.5, 0.5)$. We choose BIN_DIVISOR = 200, BIN_NUM_NORMAL = 250.

arranged in monotone increasing order in information, respectively. The figures illustrating each of these two pairs of problems also show that the corresponding p-boxes are in nested order. The informations about parameter b are in monotone increasing order in Problems 5 and 6, while the information about parameter a are in *monotone decreasing order*, thus the p-box in Problem 6 and the ones in Problem 5 do not have nested relationship.

In the three cases (i.e., consonant collection, consistent collection, and arbitrary collection) of Problems 2, 3, and 5, the figures seem to suggest some nested relationships. However the p-boxes are plotted based on the chosen intervals for a and b that present some "narrower-and-narrower" pattern in these cases. In general, p-boxes do not have nested relationship if we do not have refined information for the parameters. In fact, if we use the area of p-box to measure uncertainty, then it can be shown (see [31]) that the area of p-box is invariant with respect to the lengths of the bins and the associated probabilities in the generalized probabilistic discretization, i.e., given that the lengths and the associated probabilities of the bins are fixed, the area of the p-box is constant. Therefore, none of these three cases (consonant collection, consistent collection and arbitrary collection) provides more information about the underlying probability distributions than the others.

6 Conclusion

All simulations are approximate by nature, due to complex and inherent uncertainties in the data or the computer model being used. The uncertainty can grow or shrink during a calculation in unexpected ways. One way to quantify the uncertainty in computer simulations is to directly represent the unknowns as generalized probabilistic discretizations of PDVs and define computer arithmetic rules to accurately track the evolution of the PDVs during a calculation. The computer arithmetic should also be able to dynamically calculate the correlations among PDVs.

Based on rigorous mathematical reasoning, we have developed PDV Arithmetic that is characterized by its *complete dependency tracking* feature. Unlike other existing approaches, PDV Arithmetic provides *convergent* enclosing bounds to the solutions. In other words, PDV Arithmetic does not lose any possible solution and when the refinement is taken as infinitesimally small, PDV Arithmetic can get *exactly* all the possible solutions.

We implemented PDV Arithmetic by extending Fortran 77 to PDVFOR77 where PDVs are declared in the same manner as real variables are declared in Fortran 77. Associated with each PDV is a generalized probabilistic discretization. A PERL program parses a PDVFOR77 program to generate a standard Fortran 77 program where the PDV Arithmetic operations are performed by subroutine calls.

We demonstrated the effectiveness of the approach by comparing PDV Arithmetic directly with Monte Carlo simulations. The examples verified that a single deterministic PDV computation could accurately capture the probability distribution generated by up to many many thousands of Monte Carlo stochastic simulations.

The highlight part of PDV Arithmetic that is superior to Monte Carlo methods is its strong ability in handling imprecise probabilities. Monte Carlo methods require the prior knowledge about the probability distribution of a PDV. That means if a PDV is characterized by one of its generalized probabilistic discretizations only, Monte Carlo methods must add additional assumptions on the PDV in order to get a particular probability distribution to run the simulations. The results obtained in this way usually do not reflect the correct solutions. Our PDV Arithmetic can calculate the correct probability distribution bounds that enclose all the possible solutions and, more importantly, the bounds are sharp when the refinements are chosen sufficiently small.

PDV Arithmetic reduces to interval arithmetic when only the range intervals of PDVs are considered. The PDV Arithmetic dependency tracking feature provides a framework to give much tighter interval bounds than the classic interval approaches that do not track the correlations between the computational variables.

Because we cannot eliminate uncertainty in computer simulations, it is essential to learn far more than we now understand about uncertainty assessment and management. This knowledge is critical to our continued and expanding use of computation. As computers become ever-more-powerful tools to simulate both natural and artificial phenomena, the potential benefits of understanding uncertainty increase. PDV Arithmetic offers a new tool to assess how each source of uncertainty propagates through computations and interacts with other sources of uncertainty.

Acknowledgment

This research has been supported by the US Department of Energy, under contract W-

7405-ENG-36 and the DOE Office of Science ASCR Program in Applied Mathematics.

We also want to thank the anonymous paper reviewer for his very helpful suggestions to make this paper more accessible to the technical community.

References

[1] Alefeld G, Herzberger J. Introduction to Interval Computations, Academic Press, New York, 1983.

[2] Berleant D. Automatically verified reasoning with both intervals and probability density functions. Interval Computations 1993; 2:48-70.

[3] Berleant D, Cheng H. A software tool for automatically verified operations on intervals and probability distributions. Reliable computing 1998; 4(1):71-82.

[4] Berleant D, Goodman-Strauss C. Bounding the results of arithmetic operations on random variables of unknown dependency using intervals. Reliable Computing 1998; 4(2): 147-165.

[5] Berleant D, Xie L, Zhang J. Statool: a tool for distribution envelope determination (DEnv), an interval-based algorithm for arithmetic on random variables. Reliable Computing 2003; 9(2): 91-108.

[6] Berleant D, Zhang J. Derivation of the correlation constraint, manuscript, `http://class.ee.iastate.edu/berleant/home/Research/Pdfs/PSERC/index.htm`.

[7] Burkill JC. Functions of intervals. Proceedings of the London Mathematical Society 1924; 22:375-446.

[8] Colomba AG, Jaarsma RJ. A powerful numerical method to combine random variables. IEEE Transactions on Reliability 1980; R-29(2):126-129.

[9] Cossette H, Denuit M, Marceau É. Distributional bounds for functions of dependent risks. Bulletin of the Swiss Association of Actuaries 2002. p. 45-65.

[10] Dempster AP. Upper and lower probability induced by a multivalued mapping. Annals of Mathematical Statistics 1967a; 38(2): 325-339.

[11] Dempster AP. Upper and lower probability inferences based on a sample from a finite univariate population. Biometrika 1967b; 54(3-4): 515-528.

[12] Feller W. An Introduction to Probability Theory and Its Applications, Wiley, New York, 1957-1971.

[13] Ferson S. What Monte Carlo methods cannot do. Journal of Human and Ecological Risk Assessment 1996; 2(4): 990-1007.

[14] Ferson S. RAMAS Risk Calc 4.0: Risk Assessment with Uncertain Numbers, Lewis Press, Boca Raton, Florida, 2002.

[15] Ferson S, Hajagos J. Don't open that envelope: solutions to the Sandia problems using probability boxes. Poster, presented at Sandia, 2002 [40], `http://www.sandia.gov/epistemic/Papers/ferson.pdf`.

[16] Frank MJ, Nelson RB, Schweizer B. Best possible bounds for the distribution of a sum–a problem of Kolmogorov. Probability Theory and Related Fields 1987; 74:199-211.

[17] Gunst RF, Mason RL. Regression Analysis and its Application : a Data-Oriented Approach, M. Dekker, New York, 1980.

[18] Hyman JM. FORSIG: an extension of FORTRAN with significance arithmetic. Los Alamos National Laboratory Report, LA-9448-MS, 1982.

[19] Ingram GE, Welker EL, Herrmann CR. Designing for reliability based on probabilistic modeling using remote access

computer systems. In: Proc. 7th Reliability and Maintainability Conference, American Society of Mechanical Engineers, 1968. p. 492-500.

[20] Jaulin L, Kieffer M, Didrit O, Walter É. Applied Interval Analysis. Springer, London, Berlin, Heidelberg, New York, Barcelona, Hong Kong, Milan, Paris, Singapore, Tokyo, 2001.

[21] Kaplan S. On the method of discrete probability distributions in risk and reliability calculations, applications to seismic risk assessment. Risk Analysis 1981; 1(3): 189-196.

[22] Karloff H. Linear Programming, Birkhauser, Boston, 1991.

[23] Kazakov DA. Interval arithmetic for ADA, version 1.0, `http://www.dmitry-kazakov.de/ada/intervals.htm`.

[24] Kearfott RB. Interval computations: introduction, uses, and resources. Euromath Bulletin 1996; 2(1): 95-112.

[25] Kearfott RB. Algorithm 763: INTERVAL_ARITHMETIC: a Fortran 90 module for an interval data type. ACM Trans.Math. Software 1996; 22(4): 385-392.

[26] Kearfott RB, Kreinovich V (eds). Applications of Interval Computations. Kluwer, Dordrecht, 1996.

[27] Klir G, Wierman MJ. Uncertainty-Based Information, Elements of Generalized Information Theory, 2nd edition. Physica-Verlag, Heidelberg, New York, 1999. p 24.

[28] Korn GA, Korn TM. Mathematical Handbook for Scientists and Engineers: Definitions, Theorems, and Formulas for Reference and Review. McGraw-Hill, New York, 1968.

[29] Kreinovich V, Lakeyev A, Rohn A, Kahl P. Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht, 1997.

[30] Li W, Hyman JM. PDVFOR77: PDV Arithmetic implementation in Fortran 77, version 1.0. `http://math.lanl.gov/~liw`.

[31] Li W, Hyman JM. Theory and implementation of PDV arithmetic for uncertainty computation, in review. `http://math.lanl.gov/~liw/`.

[32] Moore RE. Interval arithmetic and automatic error analysis in digital computing. PhD thesis, Stanford University, October 1962.

[33] Moore RE. Interval Analysis. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1966.

[34] Moore RE. Methods and Applications of Interval Analysis. SIAM Studies in Applied Mathematics, Philadelphia, 1979.

[35] Moore RE. Risk analysis without Monte Carlo methods. Freiburger Intervall-Berichte 1984; 84(1): 1-48.

[36] Nelson RB. An Introduction to Copulas, Lecture Notes in Statistics 139. Springer-Verlag, 1999.

[37] Nickel KLE (ed). Interval Mathematics. Academic Press, New York, London, Toronto, Sydney, San Francisco, 1980.

[38] Oberkampf WL, Helton JC, Wojtkiewicz, SF, Joslyn CA, Ferson S. Challenge problems: uncertainty in system response given uncertain parameters. Reliability Engineering and System Safety (this issue).

[39] Regan H, Ferson S, Berleant D. Equivalence of five methods for bounding uncertainty. International Journal of Approximate Reasoning (in revision).

[40] Sandia National Laboratory. Epistemic Uncertainty Workshop, Albuquerque, August 6-7, 2002, `http://www.sandia.gov/epistemic/eup_workshop1.htm`.

[41] Shafer G. A Mathematical Theory of Evidence. Princeton University Press, Princeton, New Jersey, 1976.

[42] Sobol IM. A Primer for the Monte Carlo Method. CRC Press, Boca Raton, Ann Arbor, London, Tokyo, 1994.

[43] Statool. Software, `http://class.ee.iastate.edu/berleant/home/Research/Pdfs/versions/statool/distribution/index.htm`.

[44] Sunaga T. Theory of an Interval Algebra and its Application to Numerical Analysis. Gaukutsu Bunken Fukeyu-kai, Tokyo, 1958.

[45] Williamson, RC. Probabilistic arithmetic. Ph.D. Thesis, University of Queensland, Australia, 1989, `http://axiom.anu.edu.au/~williams/cv/pubs-standalone/pubs-standalone.html`.

[46] Williamson RC, Downs T. Calculating convolutions and dependency bounds. International Journal of Approximate Reasoning 1990; 4: 89-158.

[47] Young RC. The algebra of many valued-quantities. Math. Ann. 1931; 104:260-290.